

**Xerox
820
Information
Processor**



XEROX®

CP/M® 2.2 OPERATING SYSTEM MANUAL

820 INFORMATION PROCESSOR

Xerox is a registered trademark of Xerox Corporation

CP/M is a registered trademark of Digital Research Incorporated

Portions of this manual are reproduced by permission of Digital Research Incorporated, Pacific Grove, California

Copyright © 1981 Xerox Corporation. All rights reserved.

9R80240

Reprinted August 1981

CP/M is a registered trademark of Digital Research, Inc.

Z-80 is a registered trademark of Zilog, Inc.

Warning: This equipment generates, uses, and can radiate radio frequency energy and if not installed and used in accordance with the instructions manual, may cause interference to radio communications. As temporarily permitted by regulation it has not been tested for compliance with the limits for Class A computing devices pursuant to Subpart J of Part 15 of FCC Rules, which are designed to provide reasonable protection against such interference. Operation of this equipment in a residential area is likely to cause interference in which case the user at his own expense will be required to take whatever measures may be required to correct the interference.

TABLE of CONTENTS

GENERAL INTRODUCTION	1
FEATURES & FACILITIES	
Introduction	1
Functional Description of CP/M	3
General Command Structure	3
File References	3
Switching Disks	6
The Form of Built-In Commands	7
ERA afn cr	7
DIR afn cr	8
REN ufn1=ufn2 cr	8
SAVE n ufn cr	9
TYPE ufn cr	9
Line Editing and Output Control	11
Transient Commands	12
STAT cr	13
ASM ufn cr	16
LOAD ufn cr	17
PIP cr	18
ED ufn cr	25
SYSGEN cr	27
SUBMIT ufn parm#1 ... parm#n cr	28
DUMP ufn cr	30
MOVCPM cr	30
BDOS Error Messages	33
Operation of CP/M on the MDS	34
USER'S GUIDE	
An Overview of CP/M 2.0 Facilities	1
User Interface	3
Console Command Processor (CCP) Interface	4
STAT Enhancements	5
PIP Enhancements	8
ED Enhancements	10
The XSUB Function	11
BDOS Interface Conventions	12
CP/M 2.0 Memory Organization	27
BIOS Differences	28
ALTERATION GUIDE	
Introduction	1
First Level System Regeneration	2
Second Level System Generation	6
Sample Tetsys and Putsys Programs	10
Diskette Organization	12
The BIOS Entry Points	14
A Sample BIOS	21
A Sample Cold Start Loader	22
Reserved Locations in Page Zero	23
Disk Parameter Tables	25

The DISKDEF Macro Library	30
Sector Blocking and Deblocking	34
Appendix A	36
Appendix B	39
Appendix C	50
Appendix D	56
Appendix E	59
Appendix F	61
Appendix G	66
 INTERFACE GUIDE	
Introduction	1
Operating System Call Conventions	3
A Sample File-to-File Copy Program	29
A Sample File Dump Utility	34
A Sample Random Access Program	37
System Function Summary	46
 ASSEMBLER	
Introduction	1-1
MACRO-80 Assembler	2-1
Running MACRO-80	2-1
Command Format	2-1
Format of MACRO-80 Source Files	2-5
Expression Evaluation	2-8
Opcodes as Operands	2-10
Pseudo Operations	2-11
Macros and Block Pseudo Operations	2-25
Using Z80 Pseudo-ops	2-33
Sample Assembly	2-34
MACRO-80 Errors	2-35
Compatibility with Other Assemblies	2-36
Format of Listings	2-37
LINK-80 Linking Loader	4-1
Running LINK-80	4-1
Command Format	4-1
Format of LINK Compatible Object Files	4-7
LINK-80 Error Messages	4-10
Program Break Information	4-11
TEKDOS Operating System	A-1
TEKDOS Command Files	A-1
MACRO-80	A-1
CREF-80	A-2
LINK-80	A-2
 CONTEXT EDITOR	
Ed Tutorial	1
Introduction to ED	1
ED Operation	1
Text Transfer Functions	1
Memory Buffer Organization	5
Memory Buffer Operatin	5
Command Strings	7

Text Search and Alteration	8
Source Libraries	11
Repetitive Command Execution	12
Ed Error Conditions	13
Control Characters and Commands	14

DEBUGGING TOOL

Introduction	1
DDT Commands	3
The A (Assemble) Command	3
The D (Display) Command	4
The F (Fill) Command	4
The G (Go) Command	4
The I (Input) Command	5
The L (List) Command	6
The M (Move) Command	6
The R (Read) Command	6
The S (Set) Command	7
The T (Trace) Command	7
The U (Untrace) Command	8
The X (Examine) Command	8
Implementation Notes	9
An Example	10

DIAGNOSTICS

Diagnostics	9-1
-------------	-----

SYSTEM COMPONENTS

630 Printer	10-1
Forms Tractor	10-19
Introduction	10-23
Monitor Command Summary	10-23
Dump Command	10-24
Memory Command	10-25
Test Command	10-25
Fill Command	10-25
Copy Command	10-26
Go To Command	10-26
Read Command	10-26
Boot Command	10-27
Typewriter Command	10-27
Input Command	10-27
Output Command	10-27
Monitor Resident I/O Driver Functions	10-28
Interrupt Processing	10-28
Memory Mapped Video Display	10-28
Display Character Codes	10-31
Key Station Numbering	10-32
Key Codes	10-33
Parallel Keyboard Input	10-36
Floppy Disk Controller	10-36
Serial Input/Output	10-38
Real Time Clock	10-38
Parallel I/O Option	10-38
User Accessible Monitor Routines and Variables	10-39
Xerox 820 Subroutine Entry Points	10-39

Theory of Operation	10-42
Central Processor	10-42
Clock Generator	10-42
Reset Controller	10-42
Port Address Decoding	10-42
Disk Transfer Synchronization	10-43
CRT Display Controller	10-43
Video Scrolling	10-43
Video RAM Addressing	10-43
Video Generation	10-44
Display Blanking	10-44
64 K RAM and Bank Switching	10-44
Refresh	10-44
Bank Switching	10-44
Floppy Disk Controller, System PIO, and CTC	10-44
Floppy Disk Controller	10-44
Data Separator	10-45
Data Bus Buffering	10-45
CTC	10-45
System PIO	10-45
General Purpose PIO and SIO	10-46
SIO	10-46
Baud Rate Generator	10-46
Interrupt Structures	10-47
General Purpose PIO Strappings (J11) and Pin Assignments (J8)	10-48
CTC Strapping and I/O Assignments (J10)	10-48
Video Output Connector Pin Assignments (J7)	10-48
Serial I/O Connector Pin Assignments Channel A (J4)	10-49
Serial I/O Strapping Options for Channel A (J9)	10-49
Serial I/O Connector Pin Assignments Channel B (J3)	10-50
Keyboard Connector Pin Assignments (J2)	10-51
Disk Drive Connector Pin Assignments (J1)	10-51

INTRODUCTION

This is your CP/M Reference Manual. In it you will find complete instructions for using the CP/M operating system on your XEROX 820.

The CP/M operating system is an industry standard operating system that lets you use a variety of programs you can purchase at software houses everywhere. The instructions you'll need to load programs can be found in the CP/M Primer as well as in this reference manual.

A brief description of the 820 is given on page 1-5 of this manual. More information on the 820's single-board computer and ROM monitor commands are given in the SYSTEM COMPONENTS section of this manual (beginning on page 10-23). This information will be helpful to you as a programming aid.

To go along with the reference manual is the CP/M Primer (separate package) that you can use to learn the basics of CP/M. As you go through the CP/M Primer, you'll notice a few discrepancies between the illustrations in the Primer and what you see on your screen. For instance, the illustration may show titles vertically whereas the screen displays the titles horizontally. The MAIN difference between what you read in the Primer and what you see on the screen is how the messages are displayed.

To use this manual and the CP/M Primer, you'll need

- a Xerox 820 Information Processor (screen, keyboard, and disk drives --the printer's, optional), and
- a CP/M disk (packaged at the back of this manual), and
- a blank disk so you can make a backup copy of your CP/M disk.

We recommend you get the CP/M disk and a blank disk right now and make the backup copy of your software. That way, if anybody spills coffee on the disk, you still have your software. The instructions on the next page tell you how to prepare a disk and copy your software. The instructions are written for both 5 $\frac{1}{4}$ " and 8" disks.

We also recommend that you always remove your disks from the 820 before you turn it off. Leaving disks in the system when you power down can permanently erase information on the disks.

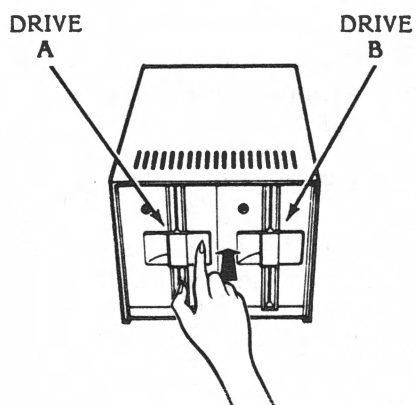
HOW TO MAKE BACK UP COPIES OF YOUR DISKS

The instructions below tell you how to load the CP/M software. Before you can copy a disk (or use the 820 to run other programs), you must load CP/M.

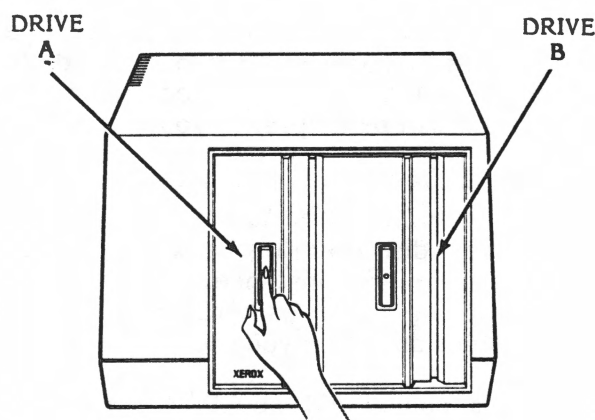
Loading the CP/M software

OPEN the disk drives (as shown below) and remove any disks

5 1/4" DISK DRIVES



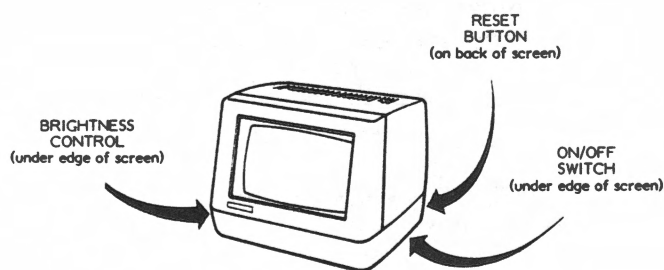
8" DISK DRIVES



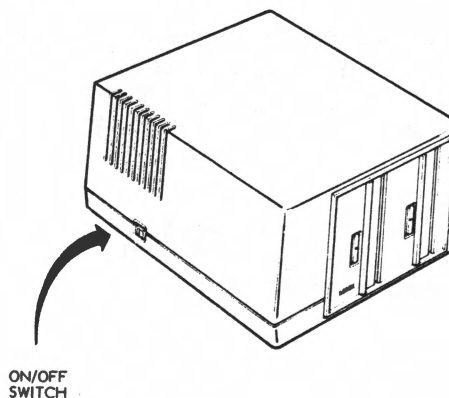
LOCATE the ON/OFF switch under the right side of the screen (shown below)

TURN the screen on, or press the RESET button if it is already ON

SCREEN



8" DISK DRIVES

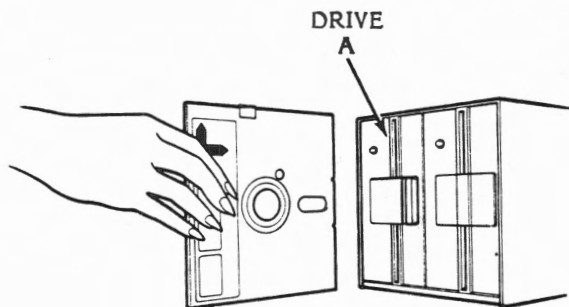


LOCATE the ON/OFF switch on the left side of the 8" disk drives, if your system has them (The 5 1/4" disk drives does not have a ON/OFF switch)

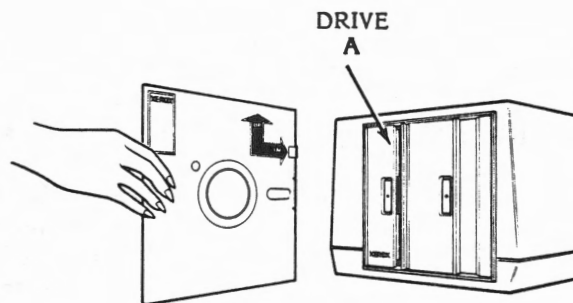
TURN the 8" disk drives on -- if your system has 8" drives, they must be turned on for your system to operate.

INSERT the CP/M disk in the A (left) drive and **CLOSE** the drive

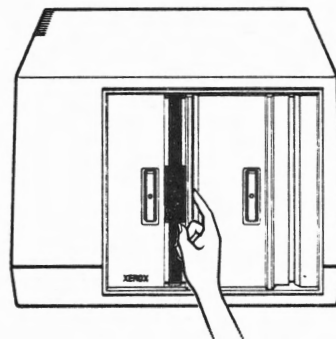
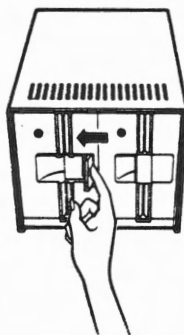
5 1/4" DISK DRIVES



8" DISK DRIVES



INSERT DISK



CLOSE DRIVE

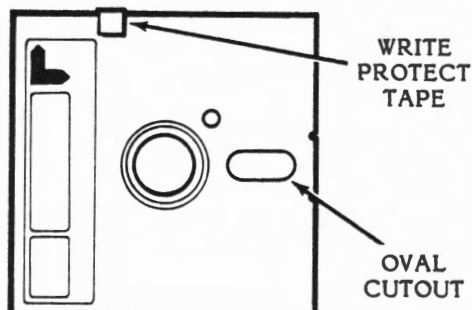
TYPE A and press **RETURN** to load CP/M into the 820's memory.

Initializing a Disk

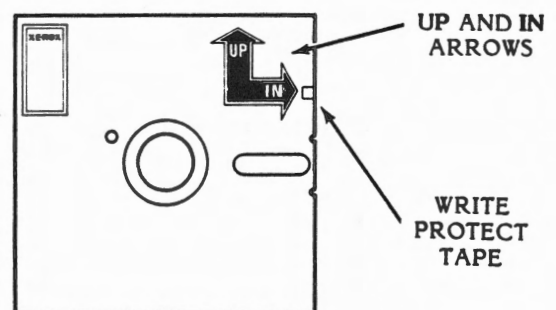
Before you can use a new disk in the 820, you must prepare it by initializing it. Follow the steps below to use the INIT program to initialize a disk. After you initialize the disk, you can copy onto it.

OBTAIN at least one new disk and take the write protect tape off of the 5 1/4" disk, or put a write protect tape on the 8" disk.
(If you don't have a new disk, a previously used one will do.)

5 1/4" DISK



8" DISK



GENERAL INTRODUCTION

1-3

7/1/81

TYPE the word **INIT** and press RETURN

WAIT for the message ENTER DISK DRIVE TO BE INITIALIZED (A OR B)

INSERT the new disk in drive B, type **B** and press RETURN twice

Wait for the message 0 FLAWED SECTORS. (If the disk has flawed sectors as indicated by a number other than 0 in front of the FLAWED SECTORS message, don't use it to copy your software - get another disk and initialize it, using steps below)

IF you had flawed sectors, initialize another disk as follows:

- Remove the disk in the **B** (right) drive
- Insert another disk in the **B** drive
- Type **B**
- Press RETURN twice
- Remove the disk after the FLAWED SECTORS message appears

WHEN all the disks are initialized:

- Press the SPACE bar
- Wait for **A >** to appear on the screen, then continue with the steps below

Copying a Disk

Now that you've initialized a disk, you are ready to copy the CP/M software.

TYPE the word **COPY** and press RETURN

Wait for a message that tells you to insert destination and source disks

BE SURE your initialized disk is in the **B** drive

PRESS the RETURN key

WAIT for the COPY COMPLETE message

REMOVE both disks. Put a label on the copy disk that says "CP/M"

PUT the original disk in the disk holder at the back of this manual

Applications Software

The instructions on page 1-2 and 1-3 told you how to load CP/M. After loading CP/M, you can run "applications" programs on the 820. The applications software may be purchased from Xerox or from other vendors.

When you use your CP/M software to run an applications program for the first time, the program may ask you to define your system. The following information will help you answer these questions.

Your 820 is configured like a Lear Siegler ADM-3A display terminal.

Your 820 has two disk drives. The 5 $\frac{1}{4}$ " disks have 40 tracks and will have 81K of available space. The 8" disks have 77 tracks and have 241K of available space.

Your software is a CP/M 2.2 operating system.

The Typewriter Feature

If your 820 has a printer, you can use it like a typewriter. Everything you type will be printed at the printer, but it will not be recorded on a disk. You do not need to have the CP/M software loaded when you use the 820 typewriter feature.

REMOVE any disks from the disk drives

TURN ON your system or press the RESET button if it is already on

INSERT paper in the printer

PRESS the T key, then press RETURN

TYPE just as you would on any typewriter

As a typewriter, the 820 uses margins of 1 and 65, with tabs set every five spaces. If you want to change these settings, use the instructions below.

SET LEFT MARGIN by pressing the SPACE bar to move to the desired position, and pressing ESC then 9

CLEAR ALL TABS by pressing ESC then 2

SET A TAB by pressing the SPACE bar to move to the desired position, and pressing ESC then 1

(This page intentionally blank)



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

AN INTRODUCTION TO CP/M FEATURES AND FACILITIES

COPYRIGHT (c) 1976, 1977, 1978

DIGITAL RESEARCH

REVISION OF JANUARY 1978

Copyright (c) 1976, 1977, 1978 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

Table of Contents

Section	Page
1. INTRODUCTION	1
2. FUNCTIONAL DESCRIPTION OF CP/M	3
2.1. General Command Structure	3
2.2. File References	3
3. SWITCHING DISKS	6
4. THE FORM OF BUILT-IN COMMANDS	7
4.1. ERA afn cr	7
4.2. DIR afn cr	8
4.3. REN ufn1=ufn2 cr	8
4.4. SAVE n ufn cr	9
4.5. TYPE ufn cr	9
5. LINE EDITING AND OUTPUT CONTROL.....	11
6. TRANSIENT COMMANDS	12
6.1. STAT cr	13
6.2. ASM ufn cr	16
6.3. LOAD ufn cr	17
6.4. PIP cr	18
6.5. ED ufn cr	25
6.6. SYSGEN cr	27
6.7. SUBMIT ufn parm#1 ... parm#n cr	28
6.8. DUMP ufn cr	30
6.9. MOVCPM cr	30
7. BDOS ERROR MESSAGES	33
8. OPERATION OF CP/M ON THE MDS	34



1. INTRODUCTION.

CP/M is a monitor control program for microcomputer system development which uses IBM-compatible flexible disks for backup storage. Using a computer mainframe based upon Intel's 8080 microcomputer, CP/M provides a general environment for program construction, storage, and editing, along with assembly and program check-out facilities. An important feature of CP/M is that it can be easily altered to execute with any computer configuration which uses an Intel 8080 (or Zilog Z-80) Central Processing Unit, and has at least 16K bytes of main memory with up to four IBM-compatible diskette drives. A detailed discussion of the modifications required for any particular hardware environment is given in the Digital Research document entitled "CP/M System Alteration Guide." Although the standard Digital Research version operates on a single-density Intel MDS 800, several different hardware manufacturers support their own input-output drivers for CP/M.

The CP/M monitor provides rapid access to programs through a comprehensive file management package. The file subsystem supports a named file structure, allowing dynamic allocation of file space as well as sequential and random file access. Using this file system, a large number of distinct programs can be stored in both source and machine executable form.

CP/M also supports a powerful context editor, Intel-compatible assembler, and debugger subsystems. Optional software includes a powerful Intel-compatible macro assembler, symbolic debugger, along with various high-level languages. When coupled with CP/M's Console Command Processor, the resulting facilities equal or excel similar large computer facilities.

CP/M is logically divided into several distinct parts:

BIOS	Basic I/O System (hardware dependent)
BDOS	Basic Disk Operating System
CCP	Console Command Processor
TPA	Transient Program Area

The BIOS provides the primitive operations necessary to access the diskette drives and to interface standard peripherals (teletype, CRT, Paper Tape Reader/Punch, and user-defined peripherals), and can be tailored by the user for any particular hardware environment by "patching" this portion of CP/M. The BDOS provides disk management by controlling one or more disk drives containing independent file directories. The BDOS implements disk allocation strategies which provide fully dynamic file construction while minimizing head movement across the disk during access. Any particular file may contain any number of records, not exceeding the size of any single disk. In a standard CP/M system, each disk can contain up to 64 distinct files. The

BDOS has entry points which include the following primitive operations which can be programmatically accessed:

SEARCH	Look for a particular disk file by name.
OPEN	Open a file for further operations.
CLOSE	Close a file after processing.
RENAME	Change the name of a particular file.
READ	Read a record from a particular file.
WRITE	Write a record onto the disk.
SELECT	Select a particular disk drive for further operations.

The CCP provides symbolic interface between the user's console and the remainder of the CP/M system. The CCP reads the console device and processes commands which include listing the file directory, printing the contents of files, and controlling the operation of transient programs, such as assemblers, editors, and debuggers. The standard commands which are available in the CCP are listed in a following section.

The last segment of CP/M is the area called the Transient Program Area (TPA). The TPA holds programs which are loaded from the disk under command of the CCP. During program editing, for example, the TPA holds the CP/M text editor machine code and data areas. Similarly, programs created under CP/M can be checked out by loading and executing these programs in the TPA.

It should be mentioned that any or all of the CP/M component subsystems can be "overlayed" by an executing program. That is, once a user's program is loaded into the TPA, the CCP, BDOS, and BIOS areas can be used as the program's data area. A "bootstrap" loader is programmatically accessible whenever the BIOS portion is not overlayed; thus, the user program need only branch to the bootstrap loader at the end of execution, and the complete CP/M monitor is reloaded from disk.

It should be reiterated that the CP/M operating system is partitioned into distinct modules, including the BIOS portion which defines the hardware environment in which CP/M is executing. Thus, the standard system can be easily modified to any non-standard environment by changing the peripheral drivers to handle the custom system.

2. FUNCTIONAL DESCRIPTION OF CP/M.

The user interacts with CP/M primarily through the CCP, which reads and interprets commands entered through the console. In general, the CCP addresses one of several disks which are online (the standard system addresses up to four different disk drives). These disk drives are labelled A, B, C, and D. A disk is "logged in" if the CCP is currently addressing the disk. In order to clearly indicate which disk is the currently logged disk, the CCP always prompts the operator with the disk name followed by the symbol ">" indicating that the CCP is ready for another command. Upon initial start up, the CP/M system is brought in from disk A, and the CCP displays the message

xxK CP/M VER m.m

where xx is the memory size (in kilobytes) which this CP/M system manages, and m.m is the CP/M version number. All CP/M systems are initially set to operate in a 16K memory space, but can be easily reconfigured to fit any memory size on the host system (see the MOVCPM transient command). Following system signon, CP/M automatically logs in disk A, prompts the user with the symbol "A>" (indicating that CP/M is currently addressing disk "A"), and waits for a command. The commands are implemented at two levels: built-in commands and transient commands.

2.1. GENERAL COMMAND STRUCTURE.

Built-in commands are a part of the CCP program itself, while transient commands are loaded into the TPA from disk and executed. The built-in commands are

ERA	Erase specified files.
DIR	List file names in the directory.
REN	Rename the specified file.
SAVE	Save memory contents in a file.
TYPE	Type the contents of a file on the logged disk.

Nearly all of the commands reference a particular file or group of files. The form of a file reference is specified below.

2.2. FILE REFERENCES.

A file reference identifies a particular file or group of files on a particular disk attached to CP/M. These file references can be either "unambiguous" (ufn) or "ambiguous" (afn). An unambiguous file reference uniquely identifies a single file, while an ambiguous file reference may be

satisfied by a number of different files.

File references consist of two parts: the primary name and the secondary name. Although the secondary name is optional, it usually is generic; that is, the secondary name "ASM," for example, is used to denote that the file is an assembly language source file, while the primary name distinguishes each particular source file. The two names are separated by a "." as shown below:

pppppppp.sss

where pppppppp represents the primary name of eight characters or less, and sss is the secondary name of no more than three characters. As mentioned above, the name

pppppppp

is also allowed and is equivalent to a secondary name consisting of three blanks. The characters used in specifying an unambiguous file reference cannot contain any of the special characters

< > . , ; : = ? * []

while all alphanumerics and remaining special characters are allowed.

An ambiguous file reference is used for directory search and pattern matching. The form of an ambiguous file reference is similar to an unambiguous reference, except the symbol "?" may be interspersed throughout the primary and secondary names. In various commands throughout CP/M, the "?" symbol matches any character of a file name in the "?" position. Thus, the ambiguous reference

X?Z.C?M

is satisfied by the unambiguous file names

XYZ.COM

and

X3Z.CAM

Note that the ambiguous reference

.

is equivalent to the ambiguous file reference

?????????.???

while

and pppppppp.*
 *.sss

are abbreviations for

 pppppppp.???
and ??????????.sss

respectively. As an example,

DIR *.*

is interpreted by the CCP as a command to list the names of all disk files in the directory, while

DIR X.Y

searches only for a file by the name X.Y Similarly, the command

DIR X?Y.C?M

causes a search for all (unambiguous) file names on the disk which satisfy this ambiguous reference.

The following file names are valid unambiguous file references:

X	XYZ	GAMMA
X.Y	XYZ.COM	GAMMA.1

As an added convenience, the programmer can generally specify the disk drive name along with the file name. In this case, the drive name is given as a letter A through Z followed by a colon (:). The specified drive is then "logged in" before the file operation occurs. Thus, the following are valid file names with disk name prefixes:

A:X.Y	B:XYZ	C:GAMMA
Z:XYZ.COM	B:X.A?M	C:*.ASM

It should also be noted that all alphabetic lower case letters in file and drive names are always translated to upper case when they are processed by the CCP.

3. SWITCHING DISKS.

The operator can switch the currently logged disk by typing the disk drive name (A, B, C, or D) followed by a colon (:) when the CCP is waiting for console input. Thus, the sequence of prompts and commands shown below might occur after the CP/M system is loaded from disk A:

16K CP/M VER 1.4

A>DIR List all files on disk A.

SAMPLE ASM

SAMPLE PRN

A>B: Switch to disk B.

B>DIR *.ASM List all "ASM" files on B.

DUMP ASM

FILES ASM

B>A: Switch back to A.

4. THE FORM OF BUILT-IN COMMANDS.

The file and device reference forms described above can now be used to fully specify the structure of the built-in commands. In the description below, assume the following abbreviations:

ufn	-	unambiguous file reference
afn	-	ambiguous file reference
cr	-	carriage return

Further, recall that the CCP always translates lower case characters to upper case characters internally. Thus, lower case alphabetic characters are treated as if they are upper case in command names and file references.

4.1 ERA afn cr

The ERA (erase) command removes files from the currently logged-in disk (i.e., the disk name currently prompted by CP/M preceding the ">"). The files which are erased are those which satisfy the ambiguous file reference afn. The following examples illustrate the use of ERA:

ERA X.Y	The file named X.Y on the currently logged disk is removed from the disk directory, and the space is returned.
ERA X.*	All files with primary name X are removed from the current disk.
ERA *.ASM	All files with secondary name ASM are removed from the current disk.
ERA X?Y.C?M	All files on the current disk which satisfy the ambiguous reference X?Y.C?M are deleted.
ERA *.*	Erase all files on the current disk (in this case the CCP prompts the console with the message "ALL FILES (Y/N)?" which requires a Y response before files are actually removed).
ERA B:*.PRN	All files on drive B which satisfy the ambiguous reference ???????.PRN are deleted, independently of the currently logged disk.

4.2. DIR afn cr

The DIR (directory) command causes the names of all files which satisfy the ambiguous file name afn to be listed at the console device. As a special case, the command

DIR

lists the files on the currently logged disk (the command "DIR" is equivalent to the command "DIR *.*"). Valid DIR commands are shown below.

DIR X.Y

DIR X?Z.C?M

DIR ??Y

Similar to other CCP commands, the afn can be preceded by a drive name. The following DIR commands cause the selected drive to be addressed before the directory search takes place.

DIR B:

DIR B:X.Y

DIR B:*.A?M

If no files can be found on the selected diskette which satisfy the directory request, then the message "NOT FOUND" is typed at the console.

4.3. REN ufn1=ufn2 cr

The REN (rename) command allows the user to change the names of files on disk. The file satisfying ufn2 is changed to ufn1. The currently logged disk is assumed to contain the file to rename (ufn1). The CCP also allows the user to type a left-directed arrow instead of the equal sign, if the user's console supports this graphic character. Examples of the REN command are

REN X.Y=Q.R

The file Q.R is changed to X.Y.

REN XYZ.COM=XYZ.XXX

The file XYZ.XXX is changed to XYZ.COM.

The operator can precede either ufn1 or ufn2 (or both) by an optional drive address. Given that ufn1 is preceded by a drive name, then ufn2 is assumed to exist on the same drive as ufn1. Similarly, if ufn2 is preceded by a drive name, then ufn1 is assumed to reside on that drive as well. If both ufn1 and ufn2 are preceded by drive names, then the same drive must be

specified in both cases. The following REN commands illustrate this format.

REN A:X.ASM = Y.ASM	The file Y.ASM is changed to X.ASM on drive A.
REN B:ZAP.BAS=ZOT.BAS	The file ZOT.BAS is changed to ZAP.BAS on drive B.
REN B:A.ASM = B:A.BAK	The file A.BAK is renamed to A.ASM on drive B.

If the file ufn1 is already present, the REN command will respond with the error "FILE EXISTS" and not perform the change. If ufn2 does not exist on the specified diskette, then the message "NOT FOUND" is printed at the console.

4.4. SAVE n ufn cr

The SAVE command places n pages (256-byte blocks) onto disk from the TPA and names this file ufn. In the CP/M distribution system, the TPA starts at 100H (hexadecimal), which is the second page of memory. Thus, if the user's program occupies the area from 100H through 2FFH, the SAVE command must specify 2 pages of memory. The machine code file can be subsequently loaded and executed. Examples are:

SAVE 3 X.COM	Copies 100H through 3FFH to X.COM.
SAVE 40 Q	Copies 100H through 28FFH to Q (note that 28 is the page count in 28FFH, and that 28H = 2*16+8 = 40 decimal).
SAVE 4 X.Y	Copies 100H through 4FFH to X.Y.

The SAVE command can also specify a disk drive in the afn portion of the command, as shown below.

SAVE 10 B:ZOT.COM	Copies 10 pages (100H through 0AFFH) to the file ZOT.COM on drive B.
-------------------	--

4.5. TYPE ufn cr

The TYPE command displays the contents of the ASCII source file ufn on the currently logged disk at the console device. Valid TYPE commands are

TYPE X.Y

TYPE X.PLM

TYPE XXX

The TYPE command expands tabs (clt-I characters), assuming tab positions are set at every eighth column. The ufn can also reference a drive name as shown below.

TYPE B:X.PRN

The file X.PRN from drive B is displayed.

5. LINE EDITING AND OUTPUT CONTROL.

The CCP allows certain line editing functions while typing command lines.

rubout	Delete and echo the last character typed at the console.
ctl-U	Delete the entire line typed at the console.
ctl-X	(Same as ctl-U)
ctl-R	Retype current command line: types a "clean line" following character deletion with rubouts.
ctl-E	Physical end of line: carriage is returned, but line is not sent until the carriage return key is depressed.
ctl-C	CP/M system reboot (warm start)
ctl-Z	End input from the console (used in PIP and ED).

The control functions ctl-P and ctl-S affect console output as shown below.

ctl-P	Copy all subsequent console output to the currently assigned list device (see the STAT command). Output is sent to both the list device and the console device until the next ctl-P is typed.
ctl-S	Stop the console output temporarily. Program execution and output continue when the next character is typed at the console (e.g., another ctl-S). This feature is used to stop output on high speed consoles, such as CRT's, in order to view a segment of output before continuing.

Note that the ctl-key sequences shown above are obtained by depressing the control and letter keys simultaneously. Further, CCP command lines can generally be up to 255 characters in length; they are not acted upon until the carriage return key is typed.

6. TRANSIENT COMMANDS.

Transient commands are loaded from the currently logged disk and executed in the TPA. The transient commands defined for execution under the CCP are shown below. Additional functions can easily be defined by the user (see the LOAD command definition).

STAT	List the number of bytes of storage remaining on the currently logged disk, provide statistical information about particular files, and display or alter device assignment.
ASM	Load the CP/M assembler and assemble the specified program from disk.
LOAD	Load the file in Intel "hex" machine code format and produce a file in machine executable form which can be loaded into the TPA (this loaded program becomes a new command under the CCP).
DDT	Load the CP/M debugger into TPA and start execution.
PIP	Load the Peripheral Interchange Program for subsequent disk file and peripheral transfer operations.
ED	Load and execute the CP/M text editor program.
SYSGEN	Create a new CP/M system diskette.
SUBMIT	Submit a file of commands for batch processing.
DUMP	Dump the contents of a file in hex.
MOVCPM	Regenerate the CP/M system for a particular memory size.

Transient commands are specified in the same manner as built-in commands, and additional commands can be easily defined by the user. As an added convenience, the transient command can be preceded by a drive name, which causes the transient to be loaded from the specified drive into the TPA for execution. Thus, the command

B:STAT

causes CP/M to temporarily "log in" drive B for the source of the STAT transient, and then return to the original logged disk for subsequent processing.

The basic transient commands are listed in detail below.

6.1. STAT cr

The STAT command provides general statistical information about file storage and device assignment. It is initiated by typing one of the following forms:

```
STAT cr
STAT "command line" cr
```

Special forms of the "command line" allow the current device assignment to be examined and altered as well. The various command lines which can be specified are shown below, with an explanation of each form shown to the right.

STAT cr

If the user types an empty command line, the STAT transient calculates the storage remaining on all active drives, and prints a message

x: R/W, SPACE: nnnK

or

x: R/O, SPACE: nnnK

for each active drive x, where R/W indicates the drive may be read or written, and R/O indicates the drive is read only (a drive becomes R/O by explicitly setting it to read only, as shown below, or by inadvertently changing diskettes without performing a warm start). The space remaining on the diskette in drive x is given in kilobytes by nnn.

STAT x: cr

If a drive name is given, then the drive is selected before the storage is computed. Thus, the command "STAT B:" could be issued while logged into drive A, resulting in the message

BYTES REMAINING ON B: nnnK

STAT afn cr

The command line can also specify a set of files to be scanned by STAT. The files which satisfy afn are listed in alphabetical order, with storage requirements for each file under the heading

```
RECS BYTS EX D:FILENAME.TYP
rrrr bbbK ee d:pppppppp.sss
```

where rrrr is the number of 128-byte records

allocated to the file, bbb is the number of kilobytes allocated to the file ($bbb = rrrr * 128 / 1024$), ee is the number of 16K extensions ($ee = bbb / 16$), d is the drive name containing the file (A...Z), pppppppp is the (up to) eight-character primary file name, and sss is the (up to) three-character secondary name. After listing the individual files, the storage usage is summarized.

STAT x:afn cr

As a convenience, the drive name can be given ahead of the afn. In this case, the specified drive is first selected, and the form "STAT afn" is executed.

STAT x:=R/O cr

This form sets the drive given by x to read-only, which remains in effect until the next warm or cold start takes place. When a disk is read-only, the message

BDOS ERR ON x: READ ONLY

will appear if there is an attempt to write to the read-only disk x. CP/M waits until a key is depressed before performing an automatic warm start (at which time the disk becomes R/W).

The STAT command also allows control over the physical to logical device assignment (see the IOBYTE function described in the manuals "CP/M Interface Guide" and "CP/M System Alteration Guide"). In general, there are four logical peripheral devices which are, at any particular instant, each assigned to one of several physical peripheral devices. The four logical devices are named:

CON:	The system console device (used by CCP for communication with the operator)
RDR:	The paper tape reader device
PUN:	The paper tape punch device
LST:	The output list device

The actual devices attached to any particular computer system are driven by subroutines in the BIOS portion of CP/M. Thus, the logical RDR: device, for example, could actually be a high speed reader, Teletype reader, or cassette tape. In order to allow some flexibility in device naming and assignment, several physical devices are defined, as shown below:

TTY:	Teletype device (slow speed console)
CRT:	Cathode ray tube device (high speed console)
BAT:	Batch processing (console is current RDR:, output goes to current LST: device)
UC1:	User-defined console
PTR:	Paper tape reader (high speed reader)
UR1:	User-defined reader #1
UR2:	User-defined reader #2
PTP:	Paper tape punch (high speed punch)
UP1:	User-defined punch #1
UP2:	User-defined punch #2
LPT:	Line printer
UL1:	User-defined list device #1

It must be emphasized that the physical device names may or may not actually correspond to devices which the names imply. That is, the PTP: device may be implemented as a cassette write operation, if the user wishes. The exact correspondence and driving subroutine is defined in the BIOS portion of CP/M. In the standard distribution version of CP/M, these devices correspond to their names on the MDS 800 development system.

The possible logical to physical device assignments can be displayed by typing

```
STAT VAL: cr
```

The STAT prints the possible values which can be taken on for each logical device:

```
CON. = TTY: CRT: BAT: UC1:
RDR: = TTY: PTR: UR1: UR2:
PUN: = TTY: PTP: UP1: UP2:
LST: = TTY: CRT: LPT: UL1:
```

In each case, the logical device shown to the left can take any of the four physical assignments shown to the right on each line. The current logical to physical mapping is displayed by typing the command

```
STAT DEV: cr
```

which produces a listing of each logical device to the left, and the current corresponding physical device to the right. For example, the list might appear as follows:

```
CON: = CRT:
RDR: = UR1:
PUN: = PTP:
LST: = TTY:
```

The current logical to physical device assignment can be changed by typing a STAT command of the form

```
STAT ld1 = pd1, ld2 = pd2 , ... , ldn = pdn cr
```

where ld1 through ldn are logical device names, and pd1 through pdn are compatible physical device names (i.e., ldi and pdi appear on the same line in the "VAL:" command shown above). The following are valid STAT commands which change the current logical to physical device assignments:

```
STAT CON:=CRT: cr
STAT PUN: = TTY:,LST:=LPT:, RDR:=TTY: cr
```

6.2. ASM ufn cr

The ASM command loads and executes the CP/M 8080 assembler. The ufn specifies a source file containing assembly language statements where the secondary name is assumed to be ASM, and thus is not specified. The following ASM commands are valid:

```
ASM X
```

```
ASM GAMMA
```

The two-pass assembler is automatically executed. If assembly errors occur during the second pass, the errors are printed at the console.

The assembler produces a file

```
x.PRN
```

where x is the primary name specified in the ASM command. The PRN file contains a listing of the source program (with imbedded tab characters if present in the source program), along with the machine code generated for each statement and diagnostic error messages, if any. The PRN file can be listed

at the console using the TYPE command, or sent to a peripheral device using PIP (see the PIP command structure below). Note also that the PRN file contains the original source program, augmented by miscellaneous assembly information in the leftmost 16 columns (program addresses and hexadecimal machine code, for example). Thus, the PRN file can serve as a backup for the original source file: if the source file is accidentally removed or destroyed, the PRN file can be edited (see the ED operator's guide) by removing the leftmost 16 characters of each line (this can be done by issuing a single editor "macro" command). The resulting file is identical to the original source file and can be renamed (REN) from PRN to ASM for subsequent editing and assembly. The file

x.HEX

is also produced which contains 8080 machine language in Intel "hex" format suitable for subsequent loading and execution (see the LOAD command). For complete details of CP/M's assembly language program, see the "CP/M Assembler Language (ASM) User's Guide."

Similar to other transient commands, the source file for assembly can be taken from an alternate disk by prefixing the assembly language file name by a disk drive name. Thus, the command

ASM B:ALPHA cr

loads the assembler from the currently logged drive and operates upon the source program ALPHA.ASM on drive B. The HEX and PRN files are also placed on drive B in this case.

6.3. LOAD ufn cr

The LOAD command reads the file ufn, which is assumed to contain "hex" format machine code, and produces a memory image file which can be subsequently executed. The file name ufn is assumed to be of the form

x.HEX

and thus only the name x need be specified in the command. The LOAD command creates a file named

x.COM

which marks it as containing machine executable code. The file is actually loaded into memory and executed when the user types the file name x immediately after the prompting character ">" printed by the CCP.

In general, the CCP reads the name x following the prompting character and looks for a built-in function name. If no function name is found, the CCP searches the system disk directory for a file by the name

x.COM

If found, the machine code is loaded into the TPA, and the program executes. Thus, the user need only LOAD a hex file once; it can be subsequently executed any number of times by simply typing the primary name. In this way, the user can "invent" new commands in the CCP. (Initialized disks contain the transient commands as COM files, which can be deleted at the user's option.) The operation can take place on an alternate drive if the file name is prefixed by a drive name. Thus,

LOAD B:BETA

brings the LOAD program into the TPA from the currently logged disk and operates upon drive B after execution begins.

It must be noted that the BETA.HEX file must contain valid Intel format hexadecimal machine code records (as produced by the ASM program, for example) which begin at 100H, the beginning of the TPA. Further, the addresses in the hex records must be in ascending order; gaps in unfilled memory regions are filled with zeroes by the LOAD command as the hex records are read. Thus, LOAD must be used only for creating CP/M standard "COM" files which operate in the TPA. Programs which occupy regions of memory other than the TPA can be loaded under DDT.

6.4. PIP cr

PIP is the CP/M Peripheral Interchange Program which implements the basic media conversion operations necessary to load, print, punch, copy, and combine disk files. The PIP program is initiated by typing one of the following forms

- (1) PIP cr
- (2) PIP "command line" cr

In both cases, PIP is loaded into the TPA and executed. In case (1), PIP reads command lines directly from the console, prompted with the "*" character, until an empty command line is typed (i.e., a single carriage return is issued by the operator). Each successive command line causes some media conversion to take place according to the rules shown below. Form (2) of the PIP command is equivalent to the first, except that the single command line given with the PIP command is automatically executed, and PIP terminates immediately with no further prompting of the console for input command lines. The form of each command line is

destination = source#1, source#2, ... , source#n cr

where "destination" is the file or peripheral device to receive the data, and

"source#1, ..., source#n" represents a series of one or more files or devices which are copied from left to right to the destination.

When multiple files are given in the command line (i.e, $n > 1$), the individual files are assumed to contain ASCII characters, with an assumed CP/M end-of-file character (ctl-Z) at the end of each file (see the O parameter to override this assumption). The equal symbol (=) can be replaced by a left-oriented arrow, if your console supports this ASCII character, to improve readability. Lower case ASCII alphabets are internally translated to upper case to be consistent with CP/M file and device name conventions. Finally, the total command line length cannot exceed 255 characters (ctl-E can be used to force a physical carriage return for lines which exceed the console width).

The destination and source elements can be unambiguous references to CP/M source files, with or without a preceding disk drive name. That is, any file can be referenced with a preceding drive name (A:, B:, C:, or D:) which defines the particular drive where the file may be obtained or stored. When the drive name is not included, the currently logged disk is assumed. Further, the destination file can also appear as one or more of the source files, in which case the source file is not altered until the entire concatenation is complete. If the destination file already exists, it is removed if the command line is properly formed (it is not removed if an error condition arises). The following command lines (with explanations to the right) are valid as input to PIP:

X = Y cr

Copy to file X from file Y, where X and Y are unambiguous file names; Y remains unchanged.

X = Y,Z cr

Concatenate files Y and Z and copy to file X, with Y and Z unchanged.

X.ASM=Y.ASM,Z.ASM,FIN.ASM cr

Create the file X.ASM from the concatenation of the Y, Z, and FIN files with type ASM.

NEW.ZOT = B:OLD.ZAP cr

Move a copy of OLD.ZAP from drive B to the currently logged disk; name the file NEW.ZOT.

B:A.U = B:B.V,A:C.W,D:X cr

Concatenate file B.V from drive B with C.W from drive A and D.X from the logged disk; create the file A.U on drive B.

For more convenient use, PIP allows abbreviated commands for transferring files between disk drives. The abbreviated forms are

PIP x:=afn cr

PIP x:=y:afn cr

PIP ufn = y: cr

PIP x:ufn = y: cr

The first form copies all files from the currently logged disk which satisfy the afn to the same file names on drive x (x = A...Z). The second form is equivalent to the first, where the source for the copy is drive y (y = A...Z). The third form is equivalent to the command "PIP ufn=y:ufn cr" which copies the file given by ufn from drive y to the file ufn on drive x. The fourth form is equivalent to the third, where the source disk is explicitly given by y.

Note that the source and destination disks must be different in all of these cases. If an afn is specified, PIP lists each ufn which satisfies the afn as it is being copied. If a file exists by the same name as the destination file, it is removed upon successful completion of the copy, and replaced by the copied file.

The following PIP commands give examples of valid disk-to-disk copy operations:

B:=*.COM cr

Copy all files which have the secondary name "COM" to drive B from the current drive.

A:=B:ZAP.* cr

Copy all files which have the primary name "ZAP" to drive A from drive B.

ZAP.ASM=B: cr

Equivalent to ZAP.ASM=B:ZAP.ASM

B:ZOT.COM=A: cr

Equivalent to B:ZOT.COM=A:ZOT.COM

B:=GAMMA.BAS cr

Same as B:GAMMA.BAS=GAMMA.BAS

B:=A:GAMMA.BAS cr

Same as B:GAMMA.BAS=A:GAMMA.BAS

PIP also allows reference to physical and logical devices which are attached to the CP/M system. The device names are the same as given under the STAT command, along with a number of specially named devices. The logical devices given in the STAT command are

CON: (console), RDR: (reader), PUN: (punch), and LST: (list)

while the physical devices are

TTY: (console, reader, punch, or list)
 CRT: (console, or list), UC1: (console)
 PTR: (reader), UR1: (reader), UR2: (reader)
 PTP: (punch), UPl: (punch), UP2: (punch)
 LPT: (list), ULL: (list)

(Note that the "BAT:" physical device is not included, since this assignment is used only to indicate that the RDR: and LST: devices are to be used for console input/output.)

The RDR, LST, PUN, and CON devices are all defined within the BIOS portion of CP/M, and thus are easily altered for any particular I/O system. (The current physical device mapping is defined by IOBYTE; see the "CP/M Interface Guide" for a discussion of this function). The destination device must be capable of receiving data (i.e., data cannot be sent to the punch), and the source devices must be capable of generating data (i.e., the LST: device cannot be read).

The additional device names which can be used in PIP commands are

NUL: Send 40 "nulls" (ASCII 0's) to the device
 (this can be issued at the end of punched output).

EOF: Send a CP/M end-of-file (ASCII ctl-Z) to the
 destination device (sent automatically at the
 end of all ASCII data transfers through PIP).

INP: Special PIP input source which can be "patched"
 into the PIP program itself: PIP gets the input
 data character-by-character by CALLing location
 103H, with data returned in location 109H (parity
 bit must be zero).

OUT: Special PIP output destination which can be
 patched into the PIP program: PIP CALLs location
 106H with data in register C for each character
 to transmit. Note that locations 109H through
 1FFH of the PIP memory image are not used and
 can be replaced by special purpose drivers using
 DDT (see the DDT operator's manual).

PRN: Same as LST:, except that tabs are expanded at
 every eighth character position, lines are
 numbered, and page ejects are inserted every 60
 lines, with an initial eject (same as [t8np]).

File and device names can be interspersed in the PIP commands. In each case, the specific device is read until end-of-file (ctl-Z for ASCII files, and a real end of file for non-ASCII disk files). Data from each device or file is concatenated from left to right until the last data source has been

read. The destination device or file is written using the data from the source files, and an end-of-file character (ctl-Z) is appended to the result for ASCII files. Note if the destination is a disk file, then a temporary file is created (\$\$\$ secondary name) which is changed to the actual file name only upon successful completion of the copy. Files with the extension "COM" are always assumed to be non-ASCII.

The copy operation can be aborted at any time by depressing any key on the keyboard (a rubout suffices). PIP will respond with the message "ABORTED" to indicate that the operation was not completed. Note that if any operation is aborted, or if an error occurs during processing, PIP removes any pending commands which were set up while using the SUBMIT command.

It should also be noted that PIP performs a special function if the destination is a disk file with type "HEX" (an Intel hex formatted machine code file), and the source is an external peripheral device, such as a paper tape reader. In this case, the PIP program checks to ensure that the source file contains a properly formed hex file, with legal hexadecimal values and checksum records. When an invalid input record is found, PIP reports an error message at the console and waits for corrective action. It is usually sufficient to open the reader and rerun a section of the tape (pull the tape back about 20 inches). When the tape is ready for the re-read, type a single carriage return at the console, and PIP will attempt another read. If the tape position cannot be properly read, simply continue the read (by typing a return following the error message), and enter the record manually with the ED program after the disk file is constructed. For convenience, PIP allows the end-of-file to be entered from the console if the source file is a RDR: device. In this case, the PIP program reads the device and monitors the keyboard. If ctl-Z is typed at the keyboard, then the read operation is terminated normally.

Valid PIP commands are shown below.

PIP LST: = X.PRN cr	Copy X.PRN to the LST device and terminate the PIP program.
PIP cr	Start PIP for a sequence of commands (PIP prompts with "*").
*CON:=X.ASM,Y.ASM,Z.ASM cr	Concatenate three ASM files and copy to the CON device.
*X.HEX=CON:;Y.HEX,PTR: cr	Create a HEX file by reading the CON (until a ctl-Z is typed), followed by data from Y.HEX, followed by data from PTR until a ctl-Z is encountered.
*cr	Single carriage return stops PIP.

PIP PUN:=NUL:,X.ASM,EOF:,NUL: cr

Send 40 nulls to the punch device; then copy the X.ASM file to the punch, followed by an end-of-file (ctl-Z) and 40 more null characters.

The user can also specify one or more PIP parameters, enclosed in left and right square brackets, separated by zero or more blanks. Each parameter affects the copy operation, and the enclosed list of parameters must immediately follow the affected file or device. Generally, each parameter can be followed by an optional decimal integer value (the S and Q parameters are exceptions). The valid PIP parameters are listed below.

- B Block mode transfer: data is buffered by PIP until an ASCII x-off character (ctl-S) is received from the source device. This allows transfer of data to a disk file from a continuous reading device, such as a cassette reader. Upon receipt of the x-off, PIP clears the disk buffers and returns for more input data. The amount of data which can be buffered is dependent upon the memory size of the host system (PIP will issue an error message if the buffers overflow).
- Dn Delete characters which extend past column n in the transfer of data to the destination from the character source. This parameter is used most often to truncate long lines which are sent to a (narrow) printer or console device.
- E Echo all transfer operations to the console as they are being performed.
- F Filter form feeds from the file. All imbedded form feeds are removed. The P parameter can be used simultaneously to insert new form feeds.
- H Hex data transfer: all data is checked for proper Intel hex file format. Non-essential characters between hex records are removed during the copy operation. The console will be prompted for corrective action in case errors occur.
- I Ignore ":00" records in the transfer of Intel hex format file (the I parameter automatically sets the H parameter).
- L Translate upper case alphabets to lower case.
- N Add line numbers to each line transferred to the destination starting at one, and incrementing by 1. Leading zeroes are suppressed, and the number is followed by a colon. If N2 is specified, then leading zeroes are included, and a tab is inserted following the number. The tab is expanded if T is

set.

- O Object file (non-ASCII) transfer: the normal CP/M end of file is ignored.
- Pn Include page ejects at every n lines (with an initial page eject). If n = 1 or is excluded altogether, page ejects occur every 60 lines. If the F parameter is used, form feed suppression takes place before the new page ejects are inserted.
- Qs↑z Quit copying from the source device or file when the string s (terminated by ctl-Z) is encountered.
- Ss↑z Start copying from the source device when the string s is encountered (terminated by ctl-Z). The S and Q parameters can be used to "abstract" a particular section of a file (such as a subroutine). The start and quit strings are always included in the copy operation.
- NOTE - the strings following the s and q parameters are translated to upper case by the CCP if form (2) of the PIP command is used. Form (1) of the PIP invocation, however, does not perform the automatic upper case translation.
- (1) PIP cr
 (2) PIP "command line" cr
- Tn Expand tabs (ctl-I characters) to every nth column during the transfer of characters to the destination from the source.
- U Translate lower case alphabets to upper case during the the copy operation.
- V Verify that data has been copied correctly by rereading after the write operation (the destination must be a disk file).
- Z Zero the parity bit on input for each ASCII character.

The following are valid PIP commands which specify parameters in the file transfer:

- | | |
|-------------------------|--|
| PIP X.ASM=B:[v] cr | Copy X.ASM from drive B to the current drive and verify that the data was properly copied. |
| PIP LPT:=X.ASM[nt8u] cr | Copy X.ASM to the LPT: device; number each line, expand tabs to every eighth column, and translate lower case alphabets to upper case. |

PIP PUN:=X.HEX[i],Y.ZOT[h] cr First copy X.HEX to the PUN: device and ignore the trailing ":00" record in X.HEX; then continue the transfer of data by reading Y.ZOT, which contains hex records, including any ":00" records which it contains.

PIP X.LIB = Y.ASM [sSUBRl:↑z qJMP L3↑z] cr Copy from the file Y.ASM into the file X.LIB. Start the copy when the string "SUBRl:" has been found, and quit copying after the string "JMP L3" is encountered.

PIP PRN:=X.ASM[p50] Send X.ASM to the LST: device, with line numbers, tabs expanded to every eighth column, and page ejects at every 50th line. Note that nt8p60 is the assumed parameter list for a PRN file; p50 overrides the default value.

6.5. ED ufn cr

The ED program is the CP/M system context editor, which allows creation and alteration of ASCII files in the CP/M environment. Complete details of operation are given the ED user's manual, "ED: a Context Editor for the CP/M Disk System." In general, ED allows the operator to create and operate upon source files which are organized as a sequence of ASCII characters, separated by end-of-line characters (a carriage-return line-feed sequence). There is no practical restriction on line length (no single line can exceed the size of the working memory), which is instead defined by the number of characters typed between cr's. The ED program has a number of commands for character string searching, replacement, and insertion, which are useful in the creation and correction of programs or text files under CP/M. Although the CP/M has a limited memory work space area (approximately 5000 characters in a 16K CP/M system), the file size which can be edited is not limited, since data is easily "paged" through this work area.

Upon initiation, ED creates the specified source file, if it does not exist, and opens the file for access. The programmer then "appends" data from the source file into the work area, if the source file already exists (see the A command), for editing. The appended data can then be displayed, altered, and written from the work area back to the disk (see the W command). Particular points in the program can be automatically paged and located by context (see the N command), allowing easy access to particular portions of a large file.

Given that the operator has typed

ED X.ASM cr

the ED program creates an intermediate work file with the name

X.\$\$\$

to hold the edited data during the ED run. Upon completion of ED, the X.ASM file (original file) is renamed to X.BAK, and the edited work file is renamed to X.ASM. Thus, the X.BAK file contains the original (unedited) file, and the X.ASM file contains the newly edited file. The operator can always return to the previous version of a file by removing the most recent version, and renaming the previous version. Suppose, for example, that the current X.ASM file was improperly edited; the sequence of CCP command shown below would reclaim the backup file.

DIR X.*	Check to see that BAK file is available.
ERA X.ASM	Erase most recent version.
REN X.ASM=X.BAK	Rename the BAK file to ASM.

Note that the operator can abort the edit at any point (reboot, power failure, ctrl-C, or Q command) without destroying the original file. In this case, the BAK file is not created, and the original file is always intact.

The ED program also allows the user to "ping-pong" the source and create backup files between two disks. The form of the ED command in this case is

ED ufn d:

where ufn is the name of a file to edit on the currently logged disk, and d is the name of an alternate drive. The ED program reads and processes the source file, and writes the new file to drive d, using the name ufn. Upon completion of processing, the original file becomes the backup file. Thus, if the operator is addressing disk A, the following command is valid:

ED X.ASM B:

which edits the file X.ASM on drive A, creating the new file X.\$\$\$ on drive B. Upon completion of a successful edit, A:X.ASM is renamed to A:X.BAK, and B:X.\$\$\$ is renamed to B:X.ASM. For user convenience, the currently logged disk becomes drive B at the end of the edit. Note that if a file by the name B:X.ASM exists before the editing begins, the message

FILE EXISTS

is printed at the console as a precaution against accidentally destroying a source file. In this case, the operator must first ERASE the existing file and then restart the edit operation.

Similar to other transient commands, editing can take place on a drive different from the currently logged disk by preceding the source file name by a drive name. Examples of valid edit requests are shown below

ED A:X.ASM

Edit the file X.ASM on drive A, with new file and backup on drive A.

ED B:X.ASM A:

Edit the file X.ASM on drive B to the temporary file X.\$\$\$ on drive A. On termination of editing, change X.ASM on drive B to X.BAK, and change X.\$\$\$ on drive A to X.ASM.

6.6. SYSGEN cr

The SYSGEN transient command allows generation of an initialized diskette containing the CP/M operating system. The SYSGEN program prompts the console for commands, with interaction as shown below.

SYSGEN cr

Initiate the SYSGEN program.

SYSGEN VERSION m.m

SYSGEN sign-on message.

SOURCE DRIVE NAME (OR RETURN TO SKIP)

Respond with the drive name (one of the letters A, B, C, or D) of the disk containing a CP/M system; usually A. If a copy of CP/M already exists in memory, due to a MOVCPM command, type a cr only. Typing a drive name x will cause the response:

SOURCE ON x THEN TYPE RETURN

Place a diskette containing the CP/M operating system on drive x (x is one of A, B, C, or D). Answer with cr when ready.

FUNCTION COMPLETE

System is copied to memory. SYSGEN will then prompt with:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

If a diskette is being initialized, place the new disk into a drive and answer with the drive name. Otherwise, type a cr and the system will reboot from drive A. Typing drive name x will cause SYSGEN to prompt

with:

DESTINATION ON x THEN TYPE RETURN Place new diskette into drive
x; type return when ready.

FUNCTION COMPLETE

New diskette is initialized
in drive x.

The "DESTINATION" prompt will be repeated until a single carriage return is typed at the console, so that more than one disk can be initialized.

Upon completion of a successful system generation, the new diskette contains the operating system, and only the built-in commands are available. A factory-fresh IBM-compatible diskette appears to CP/M as a diskette with an empty directory; therefore, the operator must copy the appropriate COM files from an existing CP/M diskette to the newly constructed diskette using the PIP transient.

The user can copy all files from an existing diskette by typing the PIP command

PIP B: = A: *.*[v] cr

which copies all files from disk drive A to disk drive B, and verifies that each file has been copied correctly. The name of each file is displayed at the console as the copy operation proceeds.

It should be noted that a SYSGEN does not destroy the files which already exist on a diskette; it results only in construction of a new operating system. Further, if a diskette is being used only on drives B through D, and will never be the source of a bootstrap operation on drive A, the SYSGEN need not take place. In fact, a new diskette needs absolutely no initialization to be used with CP/M.

6.7. SUBMIT ufn parm#1 ... parm#n cr

The SUBMIT command allows CP/M commands to be batched together for automatic processing. The ufn given in the SUBMIT command must be the filename of a file which exists on the currently logged disk, with an assumed file type of "SUB." The SUB file contains CP/M prototype commands, with possible parameter substitution. The actual parameters parm#1 ... parm#n are substituted into the prototype commands, and, if no errors occur, the file of substituted commands are processed sequentially by CP/M.

The prototype command file is created using the ED program, with interspersed "\$" parameters of the form

\$1 \$2 \$3 ... \$n

corresponding to the number of actual parameters which will be included when the file is submitted for execution. When the SUBMIT transient is executed, the actual parameters parm#1 ... parm#n are paired with the formal parameters \$1 ... \$n in the prototype commands. If the number of formal and actual parameters does not correspond, then the submit function is aborted with an error message at the console. The SUBMIT function creates a file of substituted commands with the name

\$\$\$SUB

on the logged disk. When the system reboots (at the termination of the SUBMIT), this command file is read by the CCP as a source of input, rather than the console. If the SUBMIT function is performed on any disk other than drive A, the commands are not processed until the disk is inserted into drive A and the system reboots. Further, the user can abort command processing at any time by typing a rubout when the command is read and echoed. In this case, the \$\$\$SUB file is removed, and the subsequent commands come from the console. Command processing is also aborted if the CCP detects an error in any of the commands. Programs which execute under CP/M can abort processing of command files when error conditions occur by simply erasing any existing \$\$\$SUB file.

In order to introduce dollar signs into a SUBMIT file, the user may type a "\$\$" which reduces to a single "\$" within the command file. Further, an up-arrow symbol "↑" may precede an alphabetic character x, which produces a single ctl-x character within the file.

The last command in a SUB file can initiate another SUB file, thus allowing chained batch commands.

Suppose the file ASMBL.SUB exists on disk and contains the prototype commands

```
ASM $1
DIR $1.*
ERA *.BAK
PIP $2:=$1.PRN
ERA $1.PRN
```

and the command

SUBMIT ASMBL X PRN cr

is issued by the operator. The SUBMIT program reads the ASMBL.SUB file, substituting "X" for all occurrences of \$1 and "PRN" for all occurrences of \$2, resulting in a \$\$\$SUB file containing the commands

```

ASM X
DIR X.*
ERA *.BAK
PIP PRN:=X.PRN
ERA X.PRN

```

which are executed in sequence by the CCP.

The SUBMIT function can access a SUB file which is on an alternate drive by preceding the file name by a drive name. Submitted files are only acted upon, however, when they appear on drive A. Thus, it is possible to create a submitted file on drive B which is executed at a later time when it is inserted in drive A.

6.8. DUMP ufn cr

The DUMP program types the contents of the disk file (ufn) at the console in hexadecimal form. The file contents are listed sixteen bytes at a time, with the absolute byte address listed to the left of each line in hexadecimal. Long typeouts can be aborted by pushing the rubout key during printout. (The source listing of the DUMP program is given in the "CP/M Interface Guide" as an example of a program written for the CP/M environment.)

6.9. MOVCPM cr

The MOVCPM program allows the user to reconfigure the CP/M system for any particular memory size. Two optional parameters may be used to indicate (1) the desired size of the new system and (2) the disposition of the new system at program termination. If the first parameter is omitted or a "*" is given, the MOVCPM program will reconfigure the system to its maximum size, based upon the kilobytes of contiguous RAM in the host system (starting at 0000H). If the second parameter is omitted, the system is executed, but not permanently recorded; if "*" is given, the system is left in memory, ready for a SYSGEN operation. The MOVCPM program relocates a memory image of CP/M and places this image in memory in preparation for a system generation operation. The command forms are:

MOVCPM cr

Relocate and execute CP/M for management of the current memory configuration (memory is examined for contiguous RAM, starting at 100H). Upon completion of the relocation, the new system is executed but not permanently recorded on the diskette. The system which is constructed contains a BIOS for the Intel MDS 800.

MOVCPM n cr

Create a relocated CP/M system for management of an n kilobyte system (n must be in the range 16 to 64), and execute the system, as described above.

MOVCPM * * cr

Construct a relocated memory image for the current memory configuration, but leave the memory image in memory, in preparation for a SYSGEN operation.

MOVCPM n * cr

Construct a relocated memory image for an n kilobyte memory system, and leave the memory image in preparation for a SYSGEN operation.

The command

MOVCPM * *

for example, constructs a new version of the CP/M system and leaves it in memory, ready for a SYSGEN operation. The message

READY FOR "SYSGEN" OR
"SAVE 32 CPMxx.COM"

is printed at the console upon completion, where xx is the current memory size in kilobytes. The operator can then type

SYSGEN cr

Start the system generation.

SOURCE DRIVE NAME (OR RETURN TO SKIP)

Respond with a cr to skip the CP/M read operation since the system is already in memory as a result of the previous MOVCPM operation.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Respond with B to write new system to the diskette in drive B. SYSGEN will prompt with:

DESTINATION ON B, THEN TYPE RETURN

Ready the fresh diskette on drive B and type a return when ready.

Note that if you respond with "A" rather than "B" above, the system will be written to drive A rather than B. SYSGEN will continue to type the prompt:

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

until the operator responds with a single carriage return, which stops the

SYSGEN program with a system reboot.

The user can then go through the reboot process with the old or new diskette. Instead of performing the SYSGEN operation, the user could have typed

SAVE 32 CPMxx.COM

at the completion of the MOVCPM function, which would place the CP/M memory image on the currently logged disk in a form which can be "patched." This is necessary when operating in a non-standard environment where the BIOS must be altered for a particular peripheral device configuration, as described in the "CP/M System Alteration Guide."

Valid MOVCPM commands are given below:

MOVCPM 48 cr	Construct a 48K version of CP/M and start execution.
MOVCPM 48 * cr	Construct a 48K version of CP/M in preparation for permanent recording; response is READY FOR "SYSGEN" OR "SAVE 32CPM48.COM"
MOVCPM * * cr	Construct a maximum memory version of CP/M and start execution.

It is important to note that the newly created system is serialized with the number attached to the original diskette and is subject to the conditions of the Digital Research Software Licensing Agreement.

7. BDOS ERROR MESSAGES.

There are three error situations which the Basic Disk Operating System intercepts during file processing. When one of these conditions is detected, the BDOS prints the message:

BDOS ERR ON x: error

where x is the drive name, and "error" is one of the three error messages:

BAD SECTOR
SELECT
READ ONLY

The "BAD SECTOR" message indicates that the disk controller electronics has detected an error condition in reading or writing the diskette. This condition is generally due to a malfunctioning disk controller, or an extremely worn diskette. If you find that your system reports this error more than once a month, you should check the state of your controller electronics, and the condition of your media. You may also encounter this condition in reading files generated by a controller produced by a different manufacturer. Even though controllers are claimed to be IBM-compatible, one often finds small differences in recording formats. The MDS-800 controller, for example, requires two bytes of one's following the data CRC byte, which is not required in the IBM format. As a result, diskettes generated by the Intel MDS can be read by almost all other IBM-compatible systems, while disk files generated on other manufacturer's equipment will produce the "BAD SECTOR" message when read by the MDS. In any case, recovery from this condition is accomplished by typing a `ctl-C` to reboot (this is the safest!), or a return, which simply ignores the bad sector in the file operation. Note, however, that typing a return may destroy your diskette integrity if the operation is a directory write, so make sure you have adequate backups in this case.

The "SELECT" error occurs when there is an attempt to address a drive beyond the A through D range. In this case, the value of x in the error message gives the selected drive. The system reboots following any input from the console.

The "READ ONLY" message occurs when there is an attempt to write to a diskette which has been designated as read-only in a `STAT` command, or has been set to read-only by the BDOS. In general, the operator should reboot CP/M either by using the warm start procedure (`ctl-C`) or by performing a cold start whenever the diskettes are changed. If a changed diskette is to be read but not written, BDOS allows the diskette to be changed without the warm or cold start, but internally marks the drive as read-only. The status of the drive is subsequently changed to read/write if a warm or cold start occurs. Upon issuing this message, CP/M waits for input from the console. An automatic warm start takes place following any input.

8. OPERATION OF CP/M ON THE MDS.

This section gives operating procedures for using CP/M on the Intel MDS microcomputer development system. A basic knowledge of the MDS hardware and software systems is assumed.

CP/M is initiated in essentially the same manner as Intel's ISIS operating system. The disk drives are labelled 0 through 3 on the MDS, corresponding to CP/M drives A through D, respectively. The CP/M system diskette is inserted into drive 0, and the BOOT and RESET switches are depressed in sequence. The interrupt 2 light should go on at this point. The space bar is then depressed on the device which is to be taken as the system console, and the light should go out (if it does not, then check connections and baud rates). The BOOT switch is then turned off, and the CP/M signon message should appear at the selected console device, followed by the "A>" system prompt. The user can then issue the various resident and transient commands

The CP/M system can be restarted (warm start) at any time by pushing the INT 0 switch on the front panel. The built-in Intel ROM monitor can be initiated by pushing the INT 7 switch (which generates a RST 7), except when operating under DDT, in which case the DDT program gets control instead.

Diskettes can be removed from the drives at any time, and the system can be shut down during operation without affecting data integrity. Note, however, that the user must not remove a diskette and replace it with another without rebooting the system (cold or warm start), unless the inserted diskette is "read only."

Due to hardware hang-ups or malfunctions, CP/M may type the message

BDOS ERR ON x: BAD SECTOR

where x is the drive which has a permanent error. This error may occur when drive doors are opened and closed randomly, followed by disk operations, or may be due to a diskette, drive, or controller failure. The user can optionally elect to ignore the error by typing a single return at the console. The error may produce a bad data record, requiring re-initialization of up to 128 bytes of data. The operator can reboot the CP/M system and try the operation again.

Termination of a CP/M session requires no special action, except that it is necessary to remove the diskettes before turning the power off, to avoid random transients which often make their way to the drive electronics.

It should be noted that factory-fresh IBM-compatible diskettes should be used rather than diskettes which have previously been used with any ISIS version. In particular, the ISIS "FORMAT" operation produces non-standard sector numbering throughout the diskette. This non-standard numbering seriously degrades the performance of CP/M, and will operate noticeably slower

than the distribution version. If it becomes necessary to reformat a diskette (which should not be the case for standard diskettes), a program can be written under CP/M which causes the MDS 800 controller to reformat with sequential sector numbering (1-26) on each track.

Note: "MDS 800" and "ISIS" are registered trademarks of Intel Corporation.

1. The first part of the report is a general
description of the project and its objectives.
2. The second part is a detailed description of
the methodology used in the study.
3. The third part is a description of the results
of the study.
4. The fourth part is a discussion of the results
and their implications.
5. The fifth part is a conclusion and a list of
references.



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M 2 USER'S GUIDE

COPYRIGHT (c) 1979

DIGITAL RESEARCH

Copyright

Copyright (c) 1979 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

Trademarks

CP/M is a registered trademark of Digital Research. MP/M, MAC, and SID are trademarks of Digital Research.

CP/M 2 USER'S GUIDE

Copyright (c) 1979
Digital Research, Box 579
Pacific Grove, California

1. An Overview of CP/M 2.0 Facilities	1
2. User Interface	3
3. Console Command Processor (CCP) Interface	4
4. STAT Enhancements	5
5. PIP Enhancements	8
6. ED Enhancements	10
7. The XSUB Function	11
8. BDOS Interface Conventions	12
9. CP/M 2.0 Memory Organization	27
10. BIOS Differences	28

1000 1000 1000

1000 1000 1000
1000 1000 1000
1000 1000 1000

1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000 1000 1000

1000 1000 1000 1000 1000 1000 1000 1000

1. AN OVERVIEW OF CP/M 2.0 FACILITIES.

CP/M 2.0 is a high-performance single-console operating system which uses table driven techniques to allow field reconfiguration to match a wide variety of disk capacities. All of the fundamental file restrictions are removed, while maintaining upward compatibility from previous versions of release 1. Features of CP/M 2.0 include field specification of one to sixteen logical drives, each containing up to eight megabytes. Any particular file can reach the full drive size with the capability to expand to thirty-two megabytes in future releases. The directory size can be field configured to contain any reasonable number of entries, and each file is optionally tagged with read/only and system attributes. Users of CP/M 2.0 are physically separated by user numbers, with facilities for file copy operations from one user area to another. Powerful relative-record random access functions are present in CP/M 2.0 which provide direct access to any of the 65536 records of an eight megabyte file.

All disk-dependent portions of CP/M 2.0 are placed into a BIOS-resident "disk parameter block" which is either hand coded or produced automatically using the disk definition macro library provided with CP/M 2.0. The end user need only specify the maximum number of active disks, the starting and ending sector numbers, the data allocation size, the maximum extent of the logical disk, directory size information, and reserved track values. The macros use this information to generate the appropriate tables and table references for use during CP/M 2.0 operation. Deblocking information is also provided which aids in assembly or disassembly of sector sizes which are multiples of the fundamental 128 byte data unit, and the system alteration manual includes general-purpose subroutines which use the this deblocking information to take advantage of larger sector sizes. Use of these subroutines, together with the table driven data access algorithms, make CP/M 2.0 truly a universal data management system.

File expansion is achieved by providing up to 512 logical file extents, where each logical extent contains 16K bytes of data. CP/M 2.0 is structured, however, so that as much as 128K bytes of data is addressed by a single physical extent (corresponding to a single directory entry), thus maintaining compatibility with previous versions while taking full advantage of directory space.

Random access facilities are present in CP/M 2.0 which allow immediate reference to any record of an eight megabyte file. Using CP/M's unique data organization, data blocks are only allocated when actually required and movement to a record position requires little search time. Sequential file access is upward compatible from earlier versions to the full eight megabytes, while random access compatibility stops at 512K byte files. Due to CP/M 2.0's simpler and faster random access, application programmers are encouraged to alter their programs to take full advantage of the 2.0 facilities.

Several CP/M 2.0 modules and utilities have improvements which correspond to the enhanced file system. STAT and PIP both account for file attributes and user areas, while the CCP provides a "login"

(All Information Contained Herein is Proprietary to Digital Research.)

function to change from one user area to another. The CCP also formats directory displays in a more convenient manner and accounts for both CRT and hard-copy devices in its enhanced line editing functions.

The sections below point out the individual differences between CP/M 1.4 and CP/M 2.0, with the understanding that the reader is either familiar with CP/M 1.4, or has access to the 1.4 manuals. Additional information dealing with CP/M 2.0 I/O system alteration is presented in the Digital Research manual "CP/M 2.0 Alteration Guide."

2. USER INTERFACE.

Console line processing takes CRT-type devices into account with three new control characters, shown with an asterisk in the list below (the symbol "ctl" below indicates that the control key is simultaneously depressed):

- rub/del removes and echoes last character
- ctl-C reboot when at beginning of line
- ctl-E physical end of line
- ctl-H backspace one character position*
- ctl-J (line feed) terminates current input*
- ctl-M (carriage return) terminates input
- ctl-R retype current line after new line
- ctl-U remove current line after new line
- ctl-X backspace to beginning of current line*

In particular, note that ctl-H produces the proper backspace overwrite function (ctl-H can be changed internally to another character, such as delete, through a simple single byte change). Further, the line editor keeps track of the current prompt column position so that the operator can properly align data input following a ctl-U, ctl-R, or ctl-X command.

3. CONSOLE COMMAND PROCESSOR (CCP) INTERFACE.

There are four functional differences between CP/M 1.4 and CP/M 2.0 at the console command processor (CCP) level. The CCP now displays directory information across the screen (four elements per line), the USER command is present to allow maintenance of separate files in the same directory, and the actions of the "ERA *.*" and "SAVE" commands have changed. The altered DIR format is self-explanatory, while the USER command takes the form:

USER n

where n is an integer value in the range 0 to 15. Upon cold start, the operator is automatically "logged" into user area number 0, which is compatible with standard CP/M 1.4 directories. The operator may issue the USER command at any time to move to another logical area within the same directory. Drives which are logged-in while addressing one user number are automatically active when the operator moves to another user number since a user number is simply a prefix which accesses particular directory entries on the active disks.

The active user number is maintained until changed by a subsequent USER command, or until a cold start operation when user 0 is again assumed.

Due to the fact that user numbers now tag individual directory entries, the ERA *.* command has a different effect. In version 1.4, this command can be used to erase a directory which has "garbage" information, perhaps resulting from use of a diskette under another operating system (heaven forbid!). In 2.0, however, the ERA *.* command affects only the current user number. Thus, it is necessary to write a simple utility to erase a nonsense disk (the program simply writes the hexadecimal pattern E5 throughout the disk).

The SAVE command in version 1.4 allows only a single memory save operation, with the potential of destroying the memory image due to directory operations following extent boundary changes. Version 2.0, however, does not perform directory operations in user data areas after disk writes, and thus the SAVE operation can be used any number of times without altering the memory image.

4. STAT ENHANCEMENTS.

The STAT program has a number of additional functions which allow disk parameter display, user number display, and file indicator manipulation. The command:

STAT VAL:

produces a summary of the available status commands, resulting in the output:

```
Temp R/O Disk: d:=R/O
Set Indicator: d:filename.typ $R/O $R/W $SYS $DIR
Disk Status   : DSK: d:DSK:
User Status    :USR:
Iobyte Assign:
(list of possible assignments)
```

which gives an instant summary of the possible STAT commands. The command form:

STAT d:filename.typ \$\$

where "d:" is an optional drive name, and "filename.typ" is an unambiguous or ambiguous file name, produces the output display format:

Size	Recs	Bytes	Ext	Acc
48	48	6k	1	R/O A:ED.COM
55	55	12k	1	R/O (A:PIP.COM)
65536	128	2k	2	R/W A:X.DAT

where the \$\$ parameter causes the "Size" field to be displayed (without the \$\$, the Size field is skipped, but the remaining fields are displayed). The Size field lists the virtual file size in records, while the "Recs" field sums the number of virtual records in each extent. For files constructed sequentially, the Size and Recs fields are identical. The "Bytes" field lists the actual number of bytes allocated to the corresponding file. The minimum allocation unit is determined at configuration time, and thus the number of bytes corresponds to the record count plus the remaining unused space in the last allocated block for sequential files. Random access files are given data areas only when written, so the Bytes field contains the only accurate allocation figure. In the case of random access, the Size field gives the logical end-of-file record position and the Recs field counts the logical records of each extent (each of these extents, however, may contain unallocated "holes" even though they are added into the record count). The "Ext" field counts the number of logical 16K extents allocated to the file. Unlike version 1.4, the Ext count does not necessarily correspond to the number of directory entries given to the file, since there can be up to 128K bytes (8 logical extents) directly addressed by a single directory entry, depending upon allocation size (in a special case, there are actually 256K bytes which can be directly addressed by a physical extent).

The "Acc" field gives the R/O or R/W access mode, which is changed using the commands shown below. Similarly, the parentheses

(All Information Contained Herein is Proprietary to Digital Research.)

shown around the PIP.COM file name indicate that it has the "system" indicator set, so that it will not be listed in DIR commands. The four command forms

```
STAT d:filename.typ $R/O
STAT d:filename.typ $R/W
STAT d:filename.typ $SYS
STAT d:filename.typ $DIR
```

set or reset various permanent file indicators. The R/O indicator places the file (or set of files) in a read-only status until changed by a subsequent STAT command. The R/O status is recorded in the directory with the file so that it remains R/O through intervening cold start operations. The R/W indicator places the file in a permanent read/write status. The SYS indicator attaches the system indicator to the file, while the DIR command removes the system indicator. The "filename.typ" may be ambiguous or unambiguous, but in either case, the files whose attributes are changed are listed at the console when the change occurs. The drive name denoted by "d:" is optional.

When a file is marked R/O, subsequent attempts to erase or write into the file result in a terminal BDOS message

```
Bdos Err on d: File R/O
```

The BDOS then waits for a console input before performing a subsequent warm start (a "return" is sufficient to continue). The command form

```
STAT d:DSK:
```

lists the drive characteristics of the disk named by "d:" which is in the range A:, B:, ..., P:. The drive characteristics are listed in the format:

```
  d: Drive Characteristics
65536: 128 Byte record Capacity
 8192: Kilobyte Drive Capacity
  128: 32 Byte Directory Entries
   0: Checked Directory Entries
1024: Records/ Extent
  128: Records/ Block
   58: Sectors/ Track
    2: Reserved Tracks
```

where "d:" is the selected drive, followed by the total record capacity (65536 is an 8 megabyte drive), followed by the total capacity listed in Kilobytes. The directory size is listed next, followed by the "checked" entries. The number of checked entries is usually identical to the directory size for removable media, since this mechanism is used to detect changed media during CP/M operation without an intervening warm start. For fixed media, the number is usually zero, since the media is not changed without at least a cold or warm start. The number of records per extent determines the addressing capacity of each directory entry (1024 times 128 bytes, or

(All Information Contained Herein is Proprietary to Digital Research.)

128K in the example above). The number of records per block shows the basic allocation size (in the example, 128 records/block times 128 bytes per record, or 16K bytes per block). The listing is then followed by the number of physical sectors per track and the number of reserved tracks. For logical drives which share the same physical disk, the number of reserved tracks may be quite large, since this mechanism is used to skip lower-numbered disk areas allocated to other logical disks. The command form

STAT DSK:

produces a drive characteristics table for all currently active drives. The final STAT command form is

STATUSR:

which produces a list of the user numbers which have files on the currently addressed disk. The display format is:

```
Active User : 0
Active Files: 0 1 3
```

where the first line lists the currently addressed user number, as set by the last CCP USER command, followed by a list of user numbers scanned from the current directory. In the above case, the active user number is 0 (default at cold start), with three user numbers which have active files on the current disk. The operator can subsequently examine the directories of the other user numbers by logging-in with USER 1, USER 2, or USER 3 commands, followed by a DIR command at the CCP level.

5. PIP ENHANCEMENTS.

PIP provides three new functions which account for the features of CP/M 2.0. All three functions take the form of file parameters which are enclosed in square brackets following the appropriate file names. The commands are:

Gn	Get File from User number n (n in the range 0 - 15)
W	Write over R/O files without console interrogation
R	Read system files

The G command allows one user area to receive data files from another. Assuming the operator has issued the USER 4 command at the CCP level, the PIP statement

PIP X.Y = X.Y[G2]

reads file X.Y from user number 2 into user area number 4. The command

PIP A:=A:*. *[G2]

copies all of the files from the A drive directory for user number 2 into the A drive directory of the currently logged user number. Note that to ensure file security, one cannot copy files into a different area than the one which is currently addressed by the USER command.

Note also that the PIP program itself is initially copied to a user area (so that subsequent files can be copied) using the SAVE command. The sequence of operations shown below effectively moves PIP from one user area to the next.

USER 0	login user 0
DDT PIP.COM	load PIP to memory
(note PIP size s)	
G0	return to CCP
USER 3	login user 3
SAVE s PIP.COM	

where s is the integral number of memory "pages" (256 byte segments) occupied by PIP. The number s can be determined when PIP.COM is loaded under DDT, by referring to the value under the "NEXT" display. If for example, the next available address is 1D00, then PIP.COM requires 1C hexadecimal pages (or 1 times 16 + 12 = 28 pages), and thus the value of s is 28 in the subsequent save. Once PIP is copied in this manner, it can then be copied to another disk belonging to the same user number through normal pip transfers.

Under normal operation, PIP will not overwrite a file which is set to a permanent R/O status. If attempt is made to overwrite a R/O file, the prompt

(All Information Contained Herein is Proprietary to Digital Research.)

DESTINATION FILE IS R/O, DELETE (Y/N)?

is issued. If the operator responds with the character "y" then the file is overwritten. Otherwise, the response

**** NOT DELETED ****

is issued, the file transfer is skipped, and PIP continues with the next operation in sequence. In order to avoid the prompt and response in the case of R/O file overwrite, the command line can include the W parameter, as shown below

PIP A:=B:*.COM[W]

which copies all non-system files to the A drive from the B drive, and overwrites any R/O files in the process. If the operation involves several concatenated files, the W parameter need only be included with the last file in the list, as shown in the following example

PIP A.DAT = B.DAT,F:NEW.DAT,G:OLD.DAT[W]

Files with the system attribute can be included in PIP transfers if the R parameter is included, otherwise system files are not recognized. The command line

PIP ED.COM = B:ED.COM[R]

for example, reads the ED.COM file from the B drive, even if it has been marked as a R/O and system file. The system file attributes are copied, if present.

It should be noted that downward compatibility with previous versions of CP/M is only maintained if the file does not exceed one megabyte, no file attributes are set, and the file is created by user 0. If compatibility is required with non-standard (e.g., "double density") versions of 1.4, it may be necessary to select 1.4 compatibility mode when constructing the internal disk parameter block (see the "CP/M 2.0 Alteration Guide," and refer to Section 10 which describes BIOS differences).

6. ED ENHANCEMENTS.

The CP/M standard program editor provides several new facilities in the 2.0 release. Experience has shown that most operators use the relative line numbering feature of ED, and thus the editor has the "v" (Verify Line) option set as an initial value. The operator can, of course, disable line numbering by typing the "-v" command. If you are not familiar with the ED line number mode, you may wish to refer to the Appendix in the ED user's guide, where the "v" command is described.

ED also takes file attributes into account. If the operator attempts to edit a read/only file, the message

```
** FILE IS READ/ONLY **
```

appears at the console. The file can be loaded and examined, but cannot be altered in any way. Normally, the operator simply ends the edit session, and uses STAT to change the file attribute to R/W. If the edited file has the "system" attribute set, the message

```
"SYSTEM" FILE NOT ACCESSIBLE
```

is displayed at the console, and the edit session is aborted. Again, the STAT program can be used to change the system attribute, if desired.

Finally, the insert mode ("i") command allows CRT line editing functions, as described in Section 2, above.

7. THE XSUB FUNCTION.

An additional utility program is supplied with version 2.0 of CP/M, called XSUB, which extends the power of the SUBMIT facility to include line input to programs as well as the console command processor. The XSUB command is included as the first line of your submit file and, when executed, self-relocates directly below the CCP. All subsequent submit command lines are processed by XSUB, so that programs which read buffered console input (BDOS function 10) receive their input directly from the submit file. For example, the file SAVER.SUB could contain the submit lines:

```
XSUB
DDT
I$1.HEX
R
G0
SAVE 1 $2.COM
```

with a subsequent SUBMIT command:

```
SUBMIT SAVER X Y
```

which substitutes X for \$1 and Y for \$2 in the command stream. The XSUB program loads, followed by DDT which is sent the command lines "IX.HEX" "R" and "G0" thus returning to the CCP. The final command "SAVE 1 Y.COM" is processed by the CCP.

The XSUB program remains in memory, and prints the message

(xsub active)

on each warm start operation to indicate its presence. Subsequent submit command streams do not require the XSUB, unless an intervening cold start has occurred. Note that XSUB must be loaded after DESPOOL, if both are to run simultaneously.

8. BDOS INTERFACE CONVENTIONS.

CP/M 2.0 system calls take place in exactly the same manner as earlier versions, with a call to location 0005H, function number in register C, and information address in register pair DE. Single byte values are returned in register A, with double byte values returned in HL (for reasons of compatibility, register A = L and register B = H upon return in all cases). A list of CP/M 2.0 calls is given below, with an asterisk following functions which are either new or revised from version 1.4 to 2.0. Note that a zero value is returned for out-of range function numbers.

0	System Reset	19*	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22*	Make File
4	Punch Output	23*	Rename File
5	List Output	24*	Return Login Vector
6*	Direct Console I/O	25	Return Current Disk
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Alloc)
9	Print String	28*	Write Protect Disk
10*	Read Console Buffer	29*	Get Addr(R/O Vector)
11	Get Console Status	30*	Set File Attributes
12*	Return Version Number	31*	Get Addr(Disk Parms)
13	Reset Disk System	32*	Set/Get User Code
14	Select Disk	33*	Read Random
15*	Open File	34*	Write Random
16	Close File	35*	Compute File Size
17*	Search for First	36*	Set Random Record
18*	Search for Next		

(Functions 28, 29, and 32 should be avoided in application programs to maintain upward compatibility with MP/M.) The new or revised functions are described below.

Function 6: Direct Console I/O.

Direct Console I/O is supported under CP/M 2.0 for those applications where it is necessary to avoid the BDOS console I/O operations. Programs which currently perform direct I/O through the BIOS should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or register E contains an ASCII character. If the input value is FF, then function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, then function 6 assumes that E contains a valid ASCII character which is sent to the console.

(All Information Contained Herein is Proprietary to Digital Research.)

Function 10: Read Console Buffer.

The console buffer read operation remains unchanged except that console line editing is supported, as described in Section 2. Note also that certain functions which return the carriage to the leftmost position (e.g., `ctl-X`) do so only to the column position where the prompt ended (previously, the carriage returned to the extreme left margin). This new convention makes operator data input and line correction more legible.

Function 12: Return Version Number.

Function 12 has been redefined to provide information which allows version-independent programming (this was previously the "lift head" function which returned `HL=0000` in version 1.4, but performed no operation). The value returned by function 12 is a two-byte value, with `H = 00` for the CP/M release (`H = 01` for MP/M), and `L = 00` for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, you can write application programs which provide both sequential and random access functions, with random access disabled when operating under early releases of CP/M.

In the file operations described below, DE addresses a file control block (FCB). Further, all directory operations take place in a reserved area which does not affect write buffers as was the case in version 1.4, with the exception of Search First and Search Next, where compatibility is required.

The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access, and a series of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at `005CH` can be used for random access files, since bytes `007DH`, `007EH`, and `007FH` are available for this purpose. For notational purposes, the FCB format is shown with the following fields:

```
-----
|dr|f1|f2|/ /|f8|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|rl|r2|
-----
```

```
00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35
```

where

```
dr      drive code (0 - 16)
        0 => use default drive for file
        1 => auto disk select drive A,
        2 => auto disk select drive B,
        ...
        16=> auto disk select drive P.

f1...f8  contain the file name in ASCII
        upper case, with high bit = 0

t1,t2,t3 contain the file type in ASCII
        upper case, with high bit = 0
        t1', t2', and t3' denote the
        bit of these positions,
        t1' = 1 => Read/Only file,
        t2' = 1 => SYS file, no DIR list

ex       contains the current extent number,
        normally set to 00 by the user, but
        in range 0 - 31 during file I/O

s1       reserved for internal system use

s2       reserved for internal system use, set
        to zero on call to OPEN, MAKE, SEARCH

rc       record count for extent "ex,"
        takes on values from 0 - 128

d0...dn  filled-in by CP/M, reserved for
        system use

cr       current record to read or write in
        a sequential file operation, normally
        set to zero by user

r0,r1,r2 optional random record number in the
        range 0-65535, with overflow to r2,
        r0,r1 constitute a 16-bit value with
        low byte r0, and high byte r1
```

Function 15: Open File.

The Open File operation is identical to previous definitions, with the exception that byte s2 is automatically zeroed. Note that previous versions of CP/M defined this byte as zero, but made no

(All Information Contained Herein is Proprietary to Digital Research.)

checks to assure compliance. Thus, the byte is cleared to ensure upward compatibility with the latest version, where it is required.

Function 17: Search for First.

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise a value of A equal to 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is $A * 32$ (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from fl through ex matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the dr field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the dr field is not a question mark, the s2 byte is automatically zeroed.

Function 18: Search for Next.

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

Function 19: Delete File.

The Delete File function removes files which match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the reference file or files could not be found, otherwise a value in the range 0 to 3 is returned.

(All Information Contained Herein is Proprietary to Digital Research.)

Function 22: Make File.

The Make File operation is identical to previous versions of CP/M, except that byte s2 is zeroed upon entry to the BDOS.

Function 23: Rename File.

The Actions of the file rename functions are the same as previous releases except that the value 255 is returned if the rename function is unsuccessful (the file to rename could not be found), otherwise a value in the range 0 to 3 is returned.

Function 24: Return Login Vector.

The login vector value returned by CP/M 2.0 is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A, and the high order bit of H corresponds to the sixteenth drive, labelled P. Note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

Function 28: Write Protect Current Disk.

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold or warm start operation produces the message

Bdos Err on d: R/O

Function 29: Get R/O Vector.

Function 29 returns a bit vector in register pair HL which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M which detect changed disks.

Function 30: Set File Attributes.

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2' above) can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a

(All Information Contained Herein is Proprietary to Digital Research.)

match, and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

Function 31: Get Disk Parameter Block Address.

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

Function 32: Set or Get User Code.

An application program can change or interrogate the currently active user number by calling function 32. If register E = FF hexadecimal, then the value of the current user number is returned in register A, where the value is in the range 0 to 31. If register E is not FF, then the current user number is changed to the value of E (modulo 32).

Function 33: Read Random.

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M release 2.0 does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, in version 2.0, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8 megabyte file. In order to process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an

(All Information Contained Herein is Proprietary to Digital Research.)

error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

- 01 reading unwritten data
- 02 (not returned in random mode)
- 03 cannot close current extent
- 04 seek to unwritten extent
- 05 (not returned in read mode)
- 06 seek past physical end of disk

Error code 01 and 04 occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created, which are equivalent conditions. Error 3 does not normally occur under proper system operation, but can be cleared by simply re-reading, or re-opening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is non-zero under the current 2.0 release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

Function 34: Write Random.

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Again, sequential read or write operations can commence following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Note that in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent

(All Information Contained Herein is Proprietary to Digital Research.)

switch as it does in sequential mode under either CP/M 1.4 or CP/M 2.0.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created due to directory overflow.

Function 35: Compute File Size.

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536 in version 2.0. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, only the last record of an eight megabyte file is written in random mode (i.e., record number 65535), then the virtual size is 65536 records, although only one block of data is actually allocated.

Function 36: Set Random Record.

The Set Random Record function causes the BDOS to automatically produce the random record position from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are

(All Information Contained Herein is Proprietary to Digital Research.)

involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

This section is concluded with a rather extensive, but complete example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.COM, the CCP level command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity (ok, so the program's not so brief), the only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow,

(All Information Contained Herein is Proprietary to Digital Research.)

which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

```

;*****
;*
;* sample random access program for cp/m 2.0
;*
;*****
0100          org      100h      ;base of tpa
;
0000 =        reboot    equ      0000h    ;system reboot
0005 =        bdos      equ      0005h    ;bdos entry point
;
0001 =        coninp    equ      1         ;console input function
0002 =        conout    equ      2         ;console output function
0009 =        pstring   equ      9         ;print string until '$'
000a =        rstring   equ      10        ;read console buffer
000c =        version   equ      12        ;return version number
000f =        openf     equ      15        ;file open function
0010 =        closef    equ      16        ;close function
0016 =        makef     equ      22        ;make file function
0021 =        readr     equ      33        ;read random
0022 =        writer    equ      34        ;write random
;
005c =        fcb       equ      005ch    ;default file control block
007d =        ranrec    equ      fcb+33    ;random record position
007f =        ranovf    equ      fcb+35    ;high order (overflow) byte
0080 =        buff      equ      0080h    ;buffer address
;
000d =        cr        equ      0dh      ;carriage return
000a =        lf        equ      0ah      ;line feed
;
;*****
;*
;* load SP, set-up file for random access
;*
;*****
0100 31bc0     lxi      sp,stack
;
;          version 2.0?
0103 0e0c     mvi      c,version
0105 cd050     call     bdos
0108 fe20     cpi      20h      ;version 2.0 or better?
010a d2160     jnc      versok
;          bad version, message and go back
010d 111b0     lxi      d,badver
0110 cdda0     call     print
0113 c3000     jmp      reboot
;
versok:
;          correct version for random access

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0116 0e0f      mvi      c,openf ;open default fcb
0118 115c0     lxi      d,fcbl
011b cd050     call     bdos
011e 3c        inr      a      ;err 255 becomes zero
011f c2370     jnz      ready

;
;      cannot open file, so create it
0122 0e16      mvi      c,makef
0124 115c0     lxi      d,fcbl
0127 cd050     call     bdos
012a 3c        inr      a      ;err 255 becomes zero
012b c2370     jnz      ready

;
;      cannot create file, directory full
012e 113a0     lxi      d,nospace
0131 cda00     call     print
0134 c3000     jmp      reboot ;back to ccp

;
;*****
;*
;* loop back to "ready" after each command
;*
;*****
;
ready:
;      file is ready for processing
;
0137 cde50     call     readcom ;read next command
013a 227d0     shld     ranrec ;store input record#
013d 217f0     lxi      h,ranovf
0140 3600      mvi      m,0      ;clear high byte if set
0142 fe51      cpi      'Q'      ;quit?
0144 c2560     jnz      notq

;
;      quit processing, close file
0147 0e10      mvi      c,closef
0149 115c0     lxi      d,fcbl
014c cd050     call     bdos
014f 3c        inr      a      ;err 255 becomes 0
0150 cab90     jz       error ;error message, retry
0153 c3000     jmp      reboot ;back to ccp

;
;*****
;*
;* end of quit command, process write
;*
;*****
notq:
;      not the quit command, random write?
0156 fe57      cpi      'W'
0158 c2890     jnz      notw

;
;      this is a random write, fill buffer until cr
015b 114d0     lxi      d,datmsg
015e cda00     call     print ;data prompt

```

(All Information Contained Herein is Proprietary to Digital Research.)


```

0161 0e7f      mvi      c,127      ;up to 127 characters
0163 21800     lxi      h,buff    ;destination
rloop:        ;read next character to buff
0166 c5        push     b          ;save counter
0167 e5        push     h          ;next destination
0168 cdc20     call     getchr    ;character to a
016b e1        pop      h          ;restore counter
016c c1        pop      b          ;restore next to fill
016d fe0d     cpi      cr        ;end of line?
016f ca780     jz       erloop
;            ; not end, store character
0172 77        mov      m,a
0173 23        inx      h          ;next to fill
0174 0d        dcr      c          ;counter goes down
0175 c2660     jnz      rloop     ;end of buffer?
erloop:
;            ; end of read loop, store 00
0178 3600     mvi      m,0
;
;            ; write the record to selected record number
017a 0e22     mvi      c,writer
017c 115c0     lxi      d,fcbl
017f cd050     call     bdos
0182 b7        ora      a          ;error code zero?
0183 c2b90     jnz      error      ;message if not
0186 c3370     jmp      ready     ;for another record
;
;*****
;*
;* end of write command, process read
;*
;*****
notw:
;            ; not a write command, read record?
0189 fe52     cpi      'R'
018b c2b90     jnz      error      ;skip if not
;
;            ; read random record
018e 0e21     mvi      c,readr
0190 115c0     lxi      d,fcbl
0193 cd050     call     bdos
0196 b7        ora      a          ;return code 00?
0197 c2b90     jnz      error
;
;            ; read was successful, write to console
019a cdcf0     call     crlf      ;new line
019d 0e80     mvi      c,128     ;max 128 characters
019f 21800     lxi      h,buff    ;next to get
wloop:
01a2 7e        mov      a,m        ;next character
01a3 23        inx      h          ;next to get
01a4 e67f     ani      7fh        ;mask parity
01a6 ca370     jz       ready     ;for another command if 00
01a9 c5        push     b          ;save counter
01aa e5        push     h          ;save next to get

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

01ab fe20      cpi      ;graphic?
01ad d4c80     cnc      putchar ;skip output if not
01b0 e1        pop      h
01b1 cl        pop      b
01b2 0d        dcr      c      ;count=count-1
01b3 c2a20     jnz      wloop
01b6 c3370     jmp      ready

;
;*****
;*
;* end of read command, all errors end-up here
;*
;*****
;
error:
01b9 11590     lxi      d,errmsg
01bc cdda0     call     print
01bf c3370     jmp      ready

;
;*****
;*
;* utility subroutines for console i/o
;*
;*****
getchr:
;read next console character to a
01c2 0e01     mvi      c,coninp
01c4 cd050     call     bdos
01c7 c9       ret

;
putchr:
;write character from a to console
01c8 0e02     mvi      c,conout
01ca 5f       mov      e,a      ;character to send
01cb cd050     call     bdos     ;send character
01ce c9       ret

;
crlf:
;send carriage return line feed
01cf 3e0d     mvi      a,cr      ;carriage return
01d1 cdc80     call     putchar
01d4 3e0a     mvi      a,lf      ;line feed
01d6 cdc80     call     putchar
01d9 c9       ret

;
print:
;print the buffer addressed by de until $
01da d5       push     d
01db cdcf0     call     crlf
01de d1       pop      d      ;new line
01df 0e09     mvi      c,pstring
01e1 cd050     call     bdos     ;print the string
01e4 c9       ret

;
readcom:

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

;read the next command line to the conbuf
01e5 116b0 lxi d,prompt
01e8 cdda0 call print ;command?
01eb 0e0a mvi c,rstring
01ed 117a0 lxi d,conbuf
01f0 cd050 call bdos ;read command line
; command line is present, scan it
01f3 21000 lxi h,0 ;start with 0000
01f6 117c0 lxi d,conlin;command line
01f9 1a readc: ldax d ;next command character
01fa 13 inx d ;to next command position
01fb b7 ora a ;cannot be end of command
01fc c8 rz
; not zero, numeric?
01fd d630 sui '0'
01ff fe0a cpi l0 ;carry if numeric
0201 d2130 jnc endrd
; add-in next digit
0204 29 dad h ;*2
0205 4d mov c,l
0206 44 mov b,h ;bc = value * 2
0207 29 dad h ;*4
0208 29 dad h ;*8
0209 09 dad b ;*2 + *8 = *10
020a 85 add l ;+digit
020b 6f mov l,a
020c d2f90 jnc readc ;for another char
020f 24 inr h ;overflow
0210 c3f90 jmp readc ;for another char
endrd:
; end of read, restore value in a
0213 c630 adi '0' ;command
0215 fe61 coi 'a' ;translate case?
0217 d8 rc
; lower case, mask lower case bits
0218 e65f ani 101$1111b
021a c9 ret
;
;*****
;*
;* string data area for console messages
;*
;*****
badver:
021b 536f79 db 'sorry, you need cp/m version 2$'
nospace:
023a 4e6f29 db 'no directory space$'
datmsg:
024d 547970 db 'type data: $'
errmsg:
0259 457272 db 'error, try again.$'
prompt:
026b 4e6570 db 'next command? $'
;

```

```

;*****
;*
;* fixed and variable data area
;*
;*****
027a 21 conbuf: db      conlen  ;length of console buffer
027b      consiz: ds      1      ;resulting size after read
027c      conlin: ds      32     ;length 32 buffer
0021 =    conlen equ      $-consiz
;
029c      ds      32      ;16 level stack
stack:
02bc      end

```

9. CP/M 2.0 MEMORY ORGANIZATION.

Similar to earlier versions, CP/M 2.0 is field-altered to fit various memory sizes, depending upon the host computer memory configuration. Typical base addresses for popular memory sizes are shown in the table below.

Module	20k	24k	32k	48k	64k
CCP	3400H	4400H	6400H	A400H	E400H
BDOS	3C00H	4C00H	6C00H	AC00H	EC00H
BIOS	4A00H	5A00H	7A00H	BA00H	FA00H
Top of Ram	4FFFH	5FFFH	7FFFH	BFFFH	FFFFH

The distribution disk contains a CP/M 2.0 system configured for a 20k Intel MDS-800 with standard IBM 8" floppy disk drives. The disk layout is shown below:

Sector	Track 00	Module	Track 01	Module
1	(Bootstrap Loader)		4080H	BDOS + 480H
2	3400H	CCP + 000H	4100H	BDOS + 500H
3	3480H	CCP + 080H	4180H	BDOS + 580H
4	3500H	CCP + 100H	4200H	BDOS + 600H
5	3580H	CCP + 180H	4280H	BDOS + 680H
6	3600H	CCP + 200H	4300H	BDOS + 700H
7	3680H	CCP + 280H	4380H	BDOS + 780H
8	3700H	CCP + 300H	4400H	BDOS + 800H
9	3780H	CCP + 380H	4480H	BDOS + 880H
10	3800H	CCP + 400H	4500H	BDOS + 900H
11	3880H	CCP + 480H	4580H	BDOS + 980H
12	3900H	CCP + 500H	4600H	BDOS + A00H
13	3980H	CCP + 580H	4680H	BDOS + A80H
14	3A00H	CCP + 600H	4700H	BDOS + B00H
15	3A80H	CCP + 680H	4780H	BDOS + B80H
16	3B00H	CCP + 700H	4800H	BDOS + C00H
17	3B80H	CCP + 780H	4880H	BDOS + C80H
18	3C00H	BDOS + 000H	4900H	BDOS + D00H
19	3C80H	BDOS + 080H	4980H	BDOS + D80H
20	3D00H	BDOS + 100H	4A00H	BIOS + 000H
21	3D80H	BDOS + 180H	4A80H	BIOS + 080H
22	3E00H	BDOS + 200H	4B00H	BIOS + 100H
23	3E80H	BDOS + 280H	4B80H	BIOS + 180H
24	3F00H	BDOS + 300H	4C00H	BIOS + 200H
25	3F80H	BDOS + 380H	4C80H	BIOS + 280H
26	4000H	BDOS + 400H	4D00H	BIOS + 300H

In particular, note that the CCP is at the same position on the disk, and occupies the same space as version 1.4. The BDOS portion, however, occupies one more 256-byte page and the BIOS portion extends through the remainder of track 01. Thus, the CCP is 800H (2048 decimal) bytes in length, the BDOS is E00H (3584 decimal) bytes in length, and the BIOS is up to 380H (898 decimal) bytes in length. In version 2.0, the BIOS portion contains the standard subroutines of 1.4, along with some initialized table space, as described in the following section.

(All Information Contained Herein is Proprietary to Digital Research.)

10. BIOS DIFFERENCES.

The CP/M 2.0 Basic I/O System differs only slightly in concept from its predecessors. Two new jump vector entry points are defined, a new sector translation subroutine is included, and a disk characteristics table must be defined. The skeletal form of these changes are found in the program shown below.

```
1:      org      4000h
2:      maclob   diskdef
3:      jmp      boot
4:      ;
5:      jmp      listst ;list status
6:      jmp      sectran ;sector translate
7:      disks    4
8:      ; large capacity drive
9:      bpb      equ      16*1024 ;bytes per block
10:     rpb      equ      bpb/128 ;records per block
11:     maxb     equ      65535/rpb ;max block number
12:     diskdef  0,1,58,3,bpb,maxb+1,128,0,2
13:     diskdef  1,1,58,,bpb,maxb+1,128,0,2
14:     diskdef  2,0
15:     diskdef  3,1
16:     ;
17:     boot:    ret      ;nop
18:     ;
19:     listst:   xra      a      ;nop
20:     ret
21:     ;
22:     seldsk:
23:     ;drive number in c
24:     lxi      h,0      ;0000 in hl produces select error
25:     mov      a,c      ;a is disk number 0 ... ndisks-1
26:     cpi      ndisks   ;less than ndisks?
27:     rnc      ;return with HL = 0000 if not
28:     ; proper disk number, return dpb element address
29:     mov      l,c
30:     dad      h      ;*2
31:     dad      h      ;*4
32:     dad      h      ;*8
33:     dad      h      ;*16
34:     lxi      d,dpbase
35:     dad      d      ;HL=.dpb
36:     ret
37:     ;
38:     selsec:
39:     ;sector number in c
40:     lxi      h,sector
41:     mov      m,c
42:     ret
43:     ;
44:     sectran:
45:     ;translate sector BC using table at DE
46:     xchg     ;HL = .tran
47:     dad      b      ;single precision tran
```

(All Information Contained Herein is Proprietary to Digital Research.)


```

48: ;      dad b again if double precision tran
49:      mov     1,m      ;only low byte necessary here
50: ;      fill both H and L if double precision tran
51:      ret              ;HL = ??ss
52: ;
53: sector: ds      1
54:      endef
55:      end

```

Referring to the program shown above, lines 3-6 represent the BIOS entry vector of 17 elements (version 1.4 defines only 15 jump vector elements). The last two elements provide access to the "LISTST" (List Status) entry point for DESPOOL. The use of this particular entry point is defined in the DESPOOL documentation, and is no different than the previous 1.4 release. It should be noted that the 1.4 DESPOOL program will not operate under version 2.0, but an update version will be available from Digital Research in the near future.

The "SECTTRAN" (Sector Number Translate) entry shown in the jump vector at line 6 provides access to a BIOS-resident sector translation subroutine. This mechanism allows the user to specify the sector skew factor and translation for a particular disk system, and is described below.

A macro library is shown in the listing, called DISKDEF, included on line 2, and referenced in 12-15. Although it is not necessary to use the macro library, it greatly simplifies the disk definition process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks. (See the CP/M 2.0 Alteration Guide for formulas which you can use to hand-code the tables produced by the DISKDEF library).

A BIOS disk definition consists of the following sequence of macro statements:

```

MACLIB  DISKDEF
.....
DISKS   n
DISKDEF 0,...
DISKDEF 1,...
.....
DISKDEF n-1
.....
ENDEF

```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as your BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with your system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow which define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). Note that the DISKS and DISKDEF macros generate in-line

(All Information Contained Herein is Proprietary to Digital Research.)

fixed data tables, and thus must be placed in a non-executable portion of your BIOS, typically directly following the BIOS jump vector.

The remaining portion of your BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas which are located above your BIOS.

The form of the DISKDEF macro call is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

dn	is the logical disk number, 0 to n-1
fsc	is the first physical sector number (0 or 1)
lsc	is the last sector number
skf	is the optional sector skew factor
bls	is the data allocation block size
dir	is the number of directory entries
cks	is the number of "checked" directory entries
ofs	is the track offset to logical track 00
[0]	is an optional 1.4 compatibility flag

The value "dn" is the drive number being defined with this DISKDEF macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read-only so that data is not subsequently destroyed). Normally the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system

(All Information Contained Herein is Proprietary to Digital Research.)

space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

```
DISKDEF i,j
```

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```
DISKS 4
DISKDEF 0,1,26,6,1024,243,64,64,2
DISKDEF 1,0
DISKDEF 2,0
DISKDEF 3,0

....
ENDEF
```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The definitions given in the program shown above (lines 12 through 15) provide access to the largest disks addressable by CP/M 2.0. All disks have identical parameters, except that drives 0 and 2 skip three sectors on every data access, while disks 1 and 3 access each sector in sequence as the disk revolves (there may, however, be a transparent hardware skew factor on these drives).

The DISKS macro generates n "disk header blocks," starting at address DPBASE which is a label generated by the macro. Each disk header block contains sixteen bytes, and correspond, in sequence, to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

```
DPBASE EQU $
DPE0: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3: DW XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3
```

where the DPE (disk parameter entry) labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the CP/M 2.0 Alteration Guide, but basically address the translation vector for the drive (all reference XLT0, which is the translation vector for drive 0 in the above example),

(All Information Contained Herein is Proprietary to Digital Research.)

followed by three 16-bit "scratch" addresses, followed by the directory buffer address, disk parameter block address, check vector address, and allocation vector address. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

The SELDSK function is extended somewhat in version 2.0. In particular, the selected disk number is passed to the BIOS in register C, as before, and the SELDSK subroutine performs the appropriate software or hardware actions to select the disk. Version 2.0, however, also requires the SELDSK subroutine to return the address of the selected disk parameter header (DPE0, DPE1, DPE2, or DPE3, in the above example) in register HL. If SELDSK returns the value HL = 0000H, then the BDOS assumes the disk does not exist, and prints a select error message at the terminal. Program lines 22 through 36 give a sample CP/M 2.0 SELDSK subroutine, showing only the disk parameter header address calculation.

The subroutine SECTRAN is also included in version 2.0 which performs the actual logical to physical sector translation. In earlier versions of CP/M, the sector translation process was a part of the BDOS, and set to skip six sectors between each read. Due to differing rotational speeds of various disks, the translation function has become a part of the BIOS in version 2.0. Thus, the BDOS sends sequential sector numbers to SECTRAN, starting at sector number 0. The SECTRAN subroutine uses the sequential sector number to produce a translated sector number which is returned to the BDOS. The BDOS subsequently sends the translated sector number to SELSEC before the actual read or write is performed. Note that many controllers have the capability to record the sector skew on the disk itself, and thus there is no translation necessary. In this case, the "skf" parameter is omitted in the macro call, and SECTRAN simply returns the same value which it receives. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```
XLTO: DB 1,7,13,19,25,5,11,17,23,3,9,15,21
      DB 2,8,14,20,26,6,12,18,24,4,10,16,22
```

If SECTRAN is required to translate a sector, then the following process takes place. The sector to translate is received in register pair BC. Only the C register is significant if the sector value does not exceed 255 (B = 00 in this case). Register pair DE addresses the sector translate table for this drive, determined by a previous call on SELDSK, corresponding to the first element of a disk parameter header (XLTO in the case shown above). The SECTRAN subroutine then fetches the translated sector number by adding the input sector number to the base of the translate table, to get the indexed translate table address (see lines 46, 47, and 48 in the above program). The value at this location is then returned in register L. Note that if the number of sectors exceeds 255, the translate table contains 16-bit elements whose value must be returned in HL.

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS

(All Information Contained Herein is Proprietary to Digital Research.)

which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

```
4C72 =      BEGDAT EQU $  
          (data areas)  
4DB0 =      ENDDAT EQU $  
013C =      DATSIZ EQU $-BEGDAT
```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

CP/M 2.0 is also easily adapted to disk subsystems whose sector size is a multiple of 128 bytes. Information is provided by the BDOS on sector write operations which eliminates the need for pre-read operations, thus allowing blocking and deblocking to take place at the BIOS level.

See the "CP/M 2.0 Alteration Guide" for additional details concerning tailoring your CP/M system to your particular hardware.



DIGITAL RESEARCH®

Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M 2.2 ALTERATION GUIDE

Copyright (c) 1979

DIGITAL RESEARCH

Copyright

Copyright (c) 1979 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

Trademarks

CP/M is a registered trademark of Digital Research. MP/M, MAC, and SID are trademarks of Digital Research.

CP/M 2.2 ALTERATION GUIDE

Copyright (c) 1979
Digital Research, Box 579
Pacific Grove, California

1. Introduction	1
2. First Level System Regeneration	2
3. Second Level System Generation	6
4. Sample Getsys and Putsys Programs	10
5. Diskette Organization	12
6. The BIOS Entry Points	14
7. A Sample BIOS	21
8. A Sample Cold Start Loader	22
9. Reserved Locations in Page Zero	23
10. Disk Parameter Tables	25
11. The DISKDEF Macro Library	30
12. Sector Blocking and Deblocking	34
Appendix A	36
Appendix B	39
Appendix C	50
Appendix D	56
Appendix E	59
Appendix F	61
Appendix G	66

UNIT 1: THE HISTORY OF THE UNITED STATES

1. The first part of the unit covers the early history of the United States, from the arrival of the first settlers to the founding of the nation.

2. The second part of the unit covers the period of the American Revolution, from the outbreak of the war to the signing of the Declaration of Independence.

3. The third part of the unit covers the period of the early republic, from the signing of the Constitution to the end of the War of 1812.

4. The fourth part of the unit covers the period of the Jacksonian era, from the presidency of Andrew Jackson to the end of the 1820s.

5. The fifth part of the unit covers the period of the 1830s, from the beginning of the Mexican-American War to the end of the decade.

6. The sixth part of the unit covers the period of the 1840s, from the beginning of the California Gold Rush to the end of the decade.

7. The seventh part of the unit covers the period of the 1850s, from the beginning of the Civil War to the end of the decade.

8. The eighth part of the unit covers the period of the 1860s, from the beginning of the Reconstruction era to the end of the decade.

9. The ninth part of the unit covers the period of the 1870s, from the beginning of the Gilded Age to the end of the decade.

10. The tenth part of the unit covers the period of the 1880s, from the beginning of the Progressive Era to the end of the decade.

11. The eleventh part of the unit covers the period of the 1890s, from the beginning of the Spanish-American War to the end of the decade.

12. The twelfth part of the unit covers the period of the 1900s, from the beginning of the World War I era to the end of the decade.

13. The thirteenth part of the unit covers the period of the 1910s, from the beginning of the Prohibition era to the end of the decade.

14. The fourteenth part of the unit covers the period of the 1920s, from the beginning of the Great Depression to the end of the decade.

15. The fifteenth part of the unit covers the period of the 1930s, from the beginning of the New Deal era to the end of the decade.

1. INTRODUCTION

The standard CP/M system assumes operation on an Intel MDS-800 microcomputer development system, but is designed so that the user can alter a specific set of subroutines which define the hardware operating environment. In this way, the user can produce a diskette which operates with any IBM-3741 format compatible drive controller and other peripheral devices.

Although standard CP/M 2.0 is configured for single density floppy disks, field-alteration features allow adaptation to a wide variety of disk subsystems from single drive minidisks through high-capacity "hard disk" systems. In order to simplify the following adaptation process, we assume that CP/M 2.0 will first be configured for single density floppy disks where minimal editing and debugging tools are available. If an earlier version of CP/M is available, the customizing process is eased considerably. In this latter case, you may wish to briefly review the system generation process, and skip to later sections which discuss system alteration for non-standard disk systems.

In order to achieve device independence, CP/M is separated into three distinct modules:

- BIOS - basic I/O system which is environment dependent
- BDOS - basic disk operating system which is not dependent upon the hardware configuration
- CCP - the console command processor which uses the BDOS

Of these modules, only the BIOS is dependent upon the particular hardware. That is, the user can "patch" the distribution version of CP/M to provide a new BIOS which provides a customized interface between the remaining CP/M modules and the user's own hardware system. The purpose of this document is to provide a step-by-step procedure for patching your new BIOS into CP/M.

If CP/M is being tailored to your computer system for the first time, the new BIOS requires some relatively simple software development and testing. The standard BIOS is listed in Appendix B, and can be used as a model for the customized package. A skeletal version of the BIOS is given in Appendix C which can serve as the basis for a modified BIOS. In addition to the BIOS, the user must write a simple memory loader, called GETSYS, which brings the operating system into memory. In order to patch the new BIOS into CP/M, the user must write the reverse of GETSYS, called PUTSYS, which places an altered version of CP/M back onto the diskette. PUTSYS can be derived from GETSYS by changing the disk read commands into disk write commands. Sample skeletal GETSYS and PUTSYS programs are described in Section 3, and listed in Appendix D. In order to make the CP/M system work automatically, the user must also supply a cold start loader, similar to the one provided with CP/M (listed in Appendices A and B). A skeletal form of a cold start loader is given in Appendix E which can serve as a model for your loader.

(All Information Contained Herein is Proprietary to Digital Research.)

2. FIRST LEVEL SYSTEM REGENERATION

The procedure to follow to patch the CP/M system is given below in several steps. Address references in each step are shown with a following "H" which denotes the hexadecimal radix, and are given for a 20K CP/M system. For larger CP/M systems, add a "bias" to each address which is shown with a "+b" following it, where b is equal to the memory size - 20K. Values for b in various standard memory sizes are

24K:	$b = 24K - 20K = 4K = 1000H$
32K:	$b = 32K - 20K = 12K = 3000H$
40K:	$b = 40K - 20K = 20K = 5000H$
48K:	$b = 48K - 20K = 28K = 7000H$
56K:	$b = 56K - 20K = 36K = 9000H$
62K:	$b = 62K - 20K = 42K = A800H$
64K:	$b = 64K - 20K = 44K = B000H$

Note: The standard distribution version of CP/M is set for operation within a 20K memory system. Therefore, you must first bring up the 20K CP/M system, and then configure it for your actual memory size (see Second Level System Generation).

(1) Review Section 4 and write a GETSYS program which reads the first two tracks of a diskette into memory. The data from the diskette must begin at location 3380H. Code GETSYS so that it starts at location 100H (base of the TPA), as shown in the first part of Appendix d.

(2) Test the GETSYS program by reading a blank diskette into memory, and check to see that the data has been read properly, and that the diskette has not been altered in any way by the GETSYS program.

(3) Run the GETSYS program using an initialized CP/M diskette to see if GETSYS loads CP/M starting at 3380H (the operating system actually starts 128 bytes later at 3400H).

(4) Review Section 4 and write the PUTSYS program which writes memory starting at 3380H back onto the first two tracks of the diskette. The PUTSYS program should be located at 200H, as shown in the second part of Appendix D.

(5) Test the PUTSYS program using a blank uninitialized diskette by writing a portion of memory to the first two tracks; clear memory and read it back using GETSYS. Test PUTSYS completely, since this program will be used to alter CP/M on disk.

(6) Study Sections 5, 6, and 7, along with the distribution version of the BIOS given in Appendix B, and write a simple version which performs a similar function for the customized environment. Use the program given in Appendix C as a model. Call this new BIOS by the name CBIOS (customized BIOS). Implement only the primitive disk operations on a single drive, and simple console input/output functions in this phase.

(All Information Contained Herein is Proprietary to Digital Research.)

(7) Test CBIOS completely to ensure that it properly performs console character I/O and disk reads and writes. Be especially careful to ensure that no disk write operations occur accidentally during read operations, and check that the proper track and sectors are addressed on all reads and writes. Failure to make these checks may cause destruction of the initialized CP/M system after it is patched.

(8) Referring to Figure 1 in Section 5, note that the BIOS is placed between locations 4A00H and 4FFFH. Read the CP/M system using GETSYS and replace the BIOS segment by the new CBIOS developed in step (6) and tested in step (7). This replacement is done in the memory of the machine, and will be placed on the diskette in the next step.

(9) Use PUTSYS to place the patched memory image of CP/M onto the first two tracks of a blank diskette for testing.

(10) Use GETSYS to bring the copied memory image from the test diskette back into memory at 3380H, and check to ensure that it has loaded back properly (clear memory, if possible, before the load). Upon successful load, branch to the cold start code at location 4A00H. The cold start routine will initialize page zero, then jump to the CCP at location 3400H which will call the BDOS, which will call the CBIOS. The CBIOS will be asked by the CCP to read sixteen sectors on track 2, and if successful, CP/M will type "A>", the system prompt.

When you make it this far, you are almost on the air. If you have trouble, use whatever debug facilities you have available to trace and breakpoint your CBIOS.

(11) Upon completion of step (10), CP/M has prompted the console for a command input. Test the disk write operation by typing

SAVE 1 X.COM

(recall that all commands must be followed by a carriage return).

CP/M should respond with another prompt (after several disk accesses):

A>

If it does not, debug your disk write functions and retry.

(12) Then test the directory command by typing

DIR

CP/M should respond with

A: X COM

(13) Test the erase command by typing

ERA X.COM

(All Information Contained Herein is Proprietary to Digital Research.)

CP/M should respond with the A prompt. When you make it this far, you should have an operational system which will only require a bootstrap loader to function completely.

(14) Write a bootstrap loader which is similar to GETSYS, and place it on track 0, sector 1 using PUTSYS (again using the test diskette, not the distribution diskette). See Sections 5 and 8 for more information on the bootstrap operation.

(15) Retest the new test diskette with the bootstrap loader installed by executing steps (11), (12), and (13). Upon completion of these tests, type a control-C (control and C keys simultaneously). The system should then execute a "warm start" which reboots the system, and types the A prompt.

(16) At this point, you probably have a good version of your customized CP/M system on your test diskette. Use GETSYS to load CP/M from your test diskette. Remove the test diskette, place the distribution diskette (or a legal copy) into the drive, and use PUTSYS to replace the distribution version by your customized version. Do not make this replacement if you are unsure of your patch since this step destroys the system which was sent to you from Digital Research.

(17) Load your modified CP/M system and test it by typing

DIR

CP/M should respond with a list of files which are provided on the initialized diskette. One such file should be the memory image for the debugger, called DDT.COM.

NOTE: from now on, it is important that you always reboot the CP/M system (ctl-C is sufficient) when the diskette is removed and replaced by another diskette, unless the new diskette is to be read only.

(18) Load and test the debugger by typing

DDT

(see the document "CP/M Dynamic Debugging Tool (DDT)" for operating procedures. You should take the time to become familiar with DDT, it will be your best friend in later steps.

(19) Before making further CBIOS modifications, practice using the editor (see the ED user's guide), and assembler (see the ASM user's guide). Then recode and test the GETSYS, PUTSYS, and CBIOS programs using ED, ASM, and DDT. Code and test a COPY program which does a sector-to-sector copy from one diskette to another to obtain back-up copies of the original diskette (NOTE: read your CP/M Licensing Agreement; it specifies your legal responsibilities when copying the CP/M system). Place the copyright notice

Copyright (c), 1979
Digital Research

(All Information Contained Herein is Proprietary to Digital Research.)

on each copy which is made with your COPY program.

(20) Modify your CBIOS to include the extra functions for punches, readers, signon messages, and so-forth, and add the facilities for a additional disk drives, if desired. You can make these changes with the GETSYS and PUTSYS programs which you have developed, or you can refer to the following section, which outlines CP/M facilities which will aid you in the regeneration process.

You now have a good copy of the customized CP/M system. Note that although the CBIOS portion of CP/M which you have developed belongs to you, the modified version of CP/M which you have created can be copied for your use only (again, read your Licensing Agreement), and cannot be legally copied for anyone else's use.

It should be noted that your system remains file-compatible with all other CP/M systems, (assuming media compatibility, of course) which allows transfer of non-proprietary software between users of CP/M.

3. SECOND LEVEL SYSTEM GENERATION

Now that you have the CP/M system running, you will want to configure CP/M for your memory size. In general, you will first get a memory image of CP/M with the "MOVCPM" program (system relocater) and place this memory image into a named disk file. The disk file can then be loaded, examined, patched, and replaced using the debugger, and system generation program. For further details on the operation of these programs, see the "Guide to CP/M Features and Facilities" manual.

Your CBIOS and BOOT can be modified using ED, and assembled using ASM, producing files called CBIOS.HEX and BOOT.HEX, which contain the machine code for CBIOS and BOOT in Intel hex format.

To get the memory image of CP/M into the TPA configured for the desired memory size, give the command:

```
MOVCPM xx *
```

where "xx" is the memory size in decimal K bytes (e.g., 32 for 32K). The response will be:

```
CONSTRUCTING xxK CP/M VERS 2.0  
READY FOR "SYSGEN" OR  
"SAVE 34 CPMxx.COM"
```

At this point, an image of a CP/M in the TPA configured for the requested memory size. The memory image is at location 0900H through 227FH. (i.e., The BOOT is at 0900H, the CCP is at 980H, the BDOS starts at 1180H, and the BIOS is at 1F80H.) Note that the memory image has the standard MDS-800 BIOS and BOOT on it. It is now necessary to save the memory image in a file so that you can patch your CBIOS and CBOOT into it:

```
SAVE 34 CPMxx.COM
```

The memory image created by the "MOVCPM" program is offset by a negative bias so that it loads into the free area of the TPA, and thus does not interfere with the operation of CP/M in higher memory. This memory image can be subsequently loaded under DDT and examined or changed in preparation for a new generation of the system. DDT is loaded with the memory image by typing:

```
DDT CPMxx.COM
```

Load DDT, then read the CPM image

DDT should respond with

```
NEXT PC  
2300 0100  
-
```

(The DDT prompt)

You can then use the display and disassembly commands to examine

(All Information Contained Herein is Proprietary to Digital Research.)

portions of the memory image between 900H and 227FH. Note, however, that to find any particular address within the memory image, you must apply the negative bias to the CP/M address to find the actual address. Track 00, sector 01 is loaded to location 900H (you should find the cold start loader at 900H to 97FH), track 00, sector 02 is loaded into 980H (this is the base of the CCP), and so-forth through the entire CP/M system load. In a 20K system, for example, the CCP resides at the CP/M address 3400H, but is placed into memory at 980H by the SYSGEN program. Thus, the negative bias, denoted by n, satisfies

$$3400H + n = 980H, \text{ or } n = 980H - 3400H$$

Assuming two's complement arithmetic, $n = D580H$, which can be checked by

$$3400H + D580H = 10980H = 0980H \text{ (ignoring high-order overflow).}$$

Note that for larger systems, n satisfies

$$\begin{aligned} (3400H+b) + n &= 980H, \text{ or} \\ n &= 980H - (3400H + b), \text{ or} \\ n &= D580H - b. \end{aligned}$$

The value of n for common CP/M systems is given below

memory size	bias b	negative offset n
20K	0000H	D580H - 0000H = D580H
24K	1000H	D580H - 1000H = C580H
32K	3000H	D580H - 3000H = A580H
40K	5000H	D580H - 5000H = 8580H
48K	7000H	D580H - 7000H = 6580H
56K	9000H	D580H - 9000H = 4580H
62K	A800H	D580H - A800H = 2D80H
64K	B000H	D580H - B000H = 2580H

Assume, for example, that you want to locate the address x within the memory image loaded under DDT in a 20K system. First type

Hx,n Hexadecimal sum and difference

and DDT will respond with the value of x+n (sum) and x-n (difference). The first number printed by DDT will be the actual memory address in the image where the data or code will be found. The input

H3400,D580

for example, will produce 980H as the sum, which is where the CCP is located in the memory image under DDT.

Use the L command to disassemble portions the BIOS located at $(4A00H+b)-n$ which, when you use the H command, produces an actual address of 1F80H. The disassembly command would thus be

(All Information Contained Herein is Proprietary to Digital Research.)

L1F80

It is now necessary to patch in your CBOOT and CBIOS routines. The BOOT resides at location 0900H in the memory image. If the actual load address is "n", then to calculate the bias (m) use the command:

H900,n Subtract load address from
 target address.

The second number typed in response to the command is the desired bias (m). For example, if your BOOT executes at 0080H, the command:

H900,80

will reply

0980 0880 Sum and difference in hex.

Therefore, the bias "m" would be 0880H. To read-in the BOOT, give the command:

ICBOOT.HEX Input file CBOOT.HEX

Then:

Rm Read CBOOT with a bias of
 m (=900H-n)

You may now examine your CBOOT with:

L900

We are now ready to replace the CBIOS. Examine the area at 1F80H where the original version of the CBIOS resides. Then type

ICBIOS.HEX Ready the "hex" file for loading

assume that your CBIOS is being integrated into a 20K CP/M system, and thus is originated at location 4A00H. In order to properly locate the CBIOS in the memory image under DDT, we must apply the negative bias n for a 20K system when loading the hex file. This is accomplished by typing

RD580 Read the file with bias D580H

Upon completion of the read, re-examine the area where the CBIOS has been loaded (use an "L1F80" command), to ensure that it was loaded properly. When you are satisfied that the change has been made, return from DDT using a control-C or "G0" command.

Now use SYSGEN to replace the patched memory image back onto a diskette (use a test diskette until you are sure of your patch), as shown in the following interaction

(All Information Contained Herein is Proprietary to Digital Research.)

SYSGEN
SYSGEN VERSION 2.0
SOURCE DRIVE NAME (OR RETURN TO SKIP)

Start the SYSGEN program
Sign-on message from SYSGEN
Respond with a carriage return
to skip the CP/M read operation
since the system is already in
memory.

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Respond with "B" to write the
new system to the diskette in
drive B.

DESTINATION ON B, THEN TYPE RETURN

Place a scratch diskette in
drive B, then type return.

FUNCTION COMPLETE

DESTINATION DRIVE NAME (OR RETURN TO REBOOT)

Place the scratch diskette in your drive A, and then perform a
coldstart to bring up the new CP/M system you have configured.

Test the new CP/M system, and place the Digital Research copyright
notice on the diskette, as specified in your Licensing Agreement:

Copyright (c), 1979
Digital Research

4. SAMPLE GETSYS AND PUTSYS PROGRAMS

The following program provides a framework for the GETSYS and PUTSYS programs referenced in Section 2. The READSEC and WRITESEC subroutines must be inserted by the user to read and write the specific sectors.

```

; GETSYS PROGRAM - READ TRACKS 0 AND 1 TO MEMORY AT 3380H
; REGISTER USE
; A (SCRATCH REGISTER)
; B TRACK COUNT (0, 1)
; C SECTOR COUNT (1,2,...,26)
; DE (SCRATCH REGISTER PAIR)
; HL LOAD ADDRESS
; SP SET TO STACK ADDRESS
;
START: LXI SP,3380H ;SET STACK POINTER TO SCRATCH AREA
       LXI H, 3380H ;SET BASE LOAD ADDRESS
       MVI B, 0 ;START WITH TRACK 0
RDTRK: ;READ NEXT TRACK (INITIALLY 0)
       MVI C,1 ;READ STARTING WITH SECTOR 1
RDSEC: ;READ NEXT SECTOR
       CALL READSEC ;USER-SUPPLIED SUBROUTINE
       LXI D,128 ;MOVE LOAD ADDRESS TO NEXT 1/2 PAGE
       DAD D ;HL = HL + 128
       INR C ;SECTOR = SECTOR + 1
       MOV A,C ;CHECK FOR END OF TRACK
       CPI 27
       JC RDSEC ;CARRY GENERATED IF SECTOR < 27
;
; ARRIVE HERE AT END OF TRACK, MOVE TO NEXT TRACK
       INR B
       MOV A,B ;TEST FOR LAST TRACK
       CPI 2
       JC RDTRK ;CARRY GENERATED IF TRACK < 2
;
; ARRIVE HERE AT END OF LOAD, HALT FOR NOW
       HLT
;
; USER-SUPPLIED SUBROUTINE TO READ THE DISK
READSEC:
; ENTER WITH TRACK NUMBER IN REGISTER B,
; SECTOR NUMBER IN REGISTER C, AND
; ADDRESS TO FILL IN HL
;
       PUSH B ;SAVE B AND C REGISTERS
       PUSH H ;SAVE HL REGISTERS
       .....
       perform disk read at this point, branch to
       label START if an error occurs
       .....
       POP H ;RECOVER HL
       POP B ;RECOVER B AND C REGISTERS
       RET ;BACK TO MAIN PROGRAM

END START

```

(All Information Contained Herein is Proprietary to Digital Research.)

Note that this program is assembled and listed in Appendix C for reference purposes, with an assumed origin of 100H. The hexadecimal operation codes which are listed on the left may be useful if the program has to be entered through your machine's front panel switches.

The PUTSYS program can be constructed from GETSYS by changing only a few operations in the GETSYS program given above, as shown in Appendix D. The register pair HL become the dump address (next address to write), and operations upon these registers do not change within the program. The READSEC subroutine is replaced by a WRITESEC subroutine which performs the opposite function: data from address HL is written to the track given by register B and sector given by register C. It is often useful to combine GETSYS and PUTSYS into a single program during the test and development phase, as shown in the Appendix.

5. DISKETTE ORGANIZATION

The sector allocation for the standard distribution version of CP/M is given here for reference purposes. The first sector (see table on the following page) contains an optional software boot section. Disk controllers are often set up to bring track 0, sector 1 into memory at a specific location (often location 0000H). The program in this sector, called BOOT, has the responsibility of bringing the remaining sectors into memory starting at location 3400H+b. If your controller does not have a built-in sector load, you can ignore the program in track 0, sector 1, and begin the load from track 0 sector 2 to location 3400H+b.

As an example, the Intel MDS-800 hardware cold start loader brings track 0, sector 1 into absolute address 3000H. Upon loading this sector, control transfers to location 3000H, where the bootstrap operation commences by loading the remainder of tracks 0, and all of track 1 into memory, starting at 3400H+b. The user should note that this bootstrap loader is of little use in a non-MDS environment, although it is useful to examine it since some of the boot actions will have to be duplicated in your cold start loader.

Track#	Sector#	Page#	Memory Address	CP/M Module name
00	01		(boot address)	Cold Start Loader
00	02	00	3400H+b	CCP
"	03	"	3480H+b	"
"	04	01	3500H+b	"
"	05	"	3580H+b	"
"	06	02	3600H+b	"
"	07	"	3680H+b	"
"	08	03	3700H+b	"
"	09	"	3780H+b	"
"	10	04	3800H+b	"
"	11	"	3880H+b	"
"	12	05	3900H+b	"
"	13	"	3980H+b	"
"	14	06	3A00H+b	"
"	15	"	3A80H+b	"
"	16	07	3B00H+b	"
00	17	"	3B80H+b	CCP
00	18	08	3C00H+b	BDOS
"	19	"	3C80H+b	"
"	20	09	3D00H+b	"
"	21	"	3D80H+b	"
"	22	10	3E00H+b	"
"	23	"	3E80H+b	"
"	24	11	3F00H+b	"
"	25	"	3F80H+b	"
"	26	12	4000H+b	"
01	01	"	4080H+b	"
"	02	13	4100H+b	"
"	03	"	4180H+b	"
"	04	14	4200H+b	"
"	05	"	4280H+b	"
"	06	15	4300H+b	"
"	07	"	4380H+b	"
"	08	16	4400H+b	"
"	09	"	4480H+b	"
"	10	17	4500H+b	"
"	11	"	4580H+b	"
"	12	18	4600H+b	"
"	13	"	4680H+b	"
"	14	19	4700H+b	"
"	15	"	4780H+b	"
"	16	20	4800H+b	"
"	17	"	4880H+b	"
"	18	21	4900H+b	"
01	19	"	4980H+b	BDOS
01	20	22	4A00H+b	BIOS
"	21	"	4A80H+b	"
"	23	23	4B00H+b	"
"	24	"	4B80H+b	"
"	25	24	4C00H+b	"
01	26	"	4C80H+b	BIOS
02-76	01-26		(directory and data)	

6. THE BIOS ENTRY POINTS

The entry points into the BIOS from the cold start loader and BDOS are detailed below. Entry to the BIOS is through a "jump vector" located at 4A00H+b, as shown below (see Appendices B and C, as well). The jump vector is a sequence of 17 jump instructions which send program control to the individual BIOS subroutines. The BIOS subroutines may be empty for certain functions (i.e., they may contain a single RET operation) during regeneration of CP/M, but the entries must be present in the jump vector.

The jump vector at 4A00H+b takes the form shown below, where the individual jump addresses are given to the left:

4A00H+b	JMP BOOT	; ARRIVE HERE FROM COLD START LOAD
4A03H+b	JMP WBOOT	; ARRIVE HERE FOR WARM START
4A06H+b	JMP CONST	; CHECK FOR CONSOLE CHAR READY
4A09H+b	JMP CONIN	; READ CONSOLE CHARACTER IN
4A0CH+b	JMP CONOUT	; WRITE CONSOLE CHARACTER OUT
4A0FH+b	JMP LIST	; WRITE LISTING CHARACTER OUT
4A12H+b	JMP PUNCH	; WRITE CHARACTER TO PUNCH DEVICE
4A15H+b	JMP READER	; READ READER DEVICE
4A18H+b	JMP HOME	; MOVE TO TRACK 00 ON SELECTED DISK
4A1BH+b	JMP SELDSK	; SELECT DISK DRIVE
4A1EH+b	JMP SETTRK	; SET TRACK NUMBER
4A21H+b	JMP SETSEC	; SET SECTOR NUMBER
4A24H+b	JMP SETDMA	; SET DMA ADDRESS
4A27H+b	JMP READ	; READ SELECTED SECTOR
4A2AH+b	JMP WRITE	; WRITE SELECTED SECTOR
4A2DH+b	JMP LISTST	; RETURN LIST STATUS
4A30H+b	JMP SECTAN	; SECTOR TRANSLATE SUBROUTINE

Each jump address corresponds to a particular subroutine which performs the specific function, as outlined below. There are three major divisions in the jump table: the system (re)initialization which results from calls on BOOT and WBOOT, simple character I/O performed by calls on CONST, CONIN, CONOUT, LIST, PUNCH, READER, and LISTST, and diskette I/O performed by calls on HOME, SELDSK, SETTRK, SETSEC, SETDMA, READ, WRITE, and SECTAN.

All simple character I/O operations are assumed to be performed in ASCII, upper and lower case, with high order (parity bit) set to zero. An end-of-file condition for an input device is given by an ASCII control-z (1AH). Peripheral devices are seen by CP/M as "logical" devices, and are assigned to physical devices within the BIOS.

In order to operate, the BDOS needs only the CONST, CONIN, and CONOUT subroutines (LIST, PUNCH, and READER may be used by PIP, but not the BDOS). Further, the LISTST entry is used currently only by DESPOOL, and thus, the initial version of CBIOS may have empty subroutines for the remaining ASCII devices.

(All Information Contained Herein is Proprietary to Digital Research.)

The characteristics of each device are

CONSOLE The principal interactive console which communicates with the operator, accessed through CONST, CONIN, and CONOUT. Typically, the CONSOLE is a device such as a CRT or Teletype.

LIST The principal listing device, if it exists on your system, which is usually a hard-copy device, such as a printer or Teletype.

PUNCH The principal tape punching device, if it exists, which is normally a high-speed paper tape punch or Teletype.

READER The principal tape reading device, such as a simple optical reader or Teletype.

Note that a single peripheral can be assigned as the LIST, PUNCH, and READER device simultaneously. If no peripheral device is assigned as the LIST, PUNCH, or READER device, the CBIOS created by the user may give an appropriate error message so that the system does not "hang" if the device is accessed by PIP or some other user program. Alternately, the PUNCH and LIST routines can just simply return, and the READER routine can return with a LAH (ctl-Z) in reg A to indicate immediate end-of-file.

For added flexibility, the user can optionally implement the "IOBYTE" function which allows reassignment of physical and logical devices. The IOBYTE function creates a mapping of logical to physical devices which can be altered during CP/M processing (see the STAT command). The definition of the IOBYTE function corresponds to the Intel standard as follows: a single location in memory (currently location 0003H) is maintained, called IOBYTE, which defines the logical to physical device mapping which is in effect at a particular time. The mapping is performed by splitting the IOBYTE into four distinct fields of two bits each, called the CONSOLE, READER, PUNCH, and LIST fields, as shown below:

	most significant			least significant
IOBYTE AT 0003H	LIST	PUNCH	READER	CONSOLE
	bits 6,7	bits 4,5	bits 2,3	bits 0,1

The value in each field can be in the range 0-3, defining the assigned source or destination of each logical device. The values which can be assigned to each field are given below

CONSOLE field (bits 0,1)

- 0 - console is assigned to the console printer device (TTY:)
- 1 - console is assigned to the CRT device (CRT:)
- 2 - batch mode: use the READER as the CONSOLE input,
and the LIST device as the CONSOLE output (BAT:)
- 3 - user defined console device (UC1:)

READER field (bits 2,3)

- 0 - READER is the Teletype device (TTY:)
- 1 - READER is the high-speed reader device (RDR:)
- 2 - user defined reader # 1 (UR1:)
- 3 - user defined reader # 2 (UR2:)

PUNCH field (bits 4,5)

- 0 - PUNCH is the Teletype device (TTY:)
- 1 - PUNCH is the high speed punch device (PUN:)
- 2 - user defined punch # 1 (UP1:)
- 3 - user defined punch # 2 (UP2:)

LIST field (bits 6,7)

- 0 - LIST is the Teletype device (TTY:)
- 1 - LIST is the CRT device (CRT:)
- 2 - LIST is the line printer device (LPT:)
- 3 - user defined list device (UL1:)

Note again that the implementation of the IOBYTE is optional, and affects only the organization of your CBIOS. No CP/M systems use the IOBYTE (although they tolerate the existence of the IOBYTE at location 0003H), except for PIP which allows access to the physical devices, and STAT which allows logical-physical assignments to be made and/or displayed (for more information, see the "CP/M Features and Facilities Guide"). In any case, the IOBYTE implementation should be omitted until your basic CBIOS is fully implemented and tested; then add the IOBYTE to increase your facilities.

Disk I/O is always performed through a sequence of calls on the various disk access subroutines which set up the disk number to access, the track and sector on a particular disk, and the direct memory access (DMA) address involved in the I/O operation. After all these parameters have been set up, a call is made to the READ or WRITE function to perform the actual I/O operation. Note that there is often a single call to SELDSK to select a disk drive, followed by a number of read or write operations to the selected disk before selecting another drive for subsequent operations. Similarly, there may be a single call to set the DMA address, followed by several calls which read or write from the selected DMA address before the DMA address is changed. The track and sector subroutines are always called before the READ or WRITE operations are performed.

Note that the READ and WRITE routines should perform several retries (10 is standard) before reporting the error condition to the BDOS. If the error condition is returned to the BDOS, it will report the error to the user. The HOME subroutine may or may not actually perform the track 00 seek, depending upon your controller characteristics; the important point is that track 00 has been selected for the next operation, and is often treated in exactly the same manner as SETTRK with a parameter of 00.

The exact responsibilities of each entry point subroutine are given below:

BOOT The BOOT entry point gets control from the cold start loader and is responsible for basic system initialization, including sending a signon message (which can be omitted in the first version). If the IOBYTE function is implemented, it must be set at this point. The various system parameters which are set by the WBOOT entry point must be initialized, and control is transferred to the CCP at 3400H+b for further processing. Note that reg C must be set to zero to select drive A.

WBOOT The WBOOT entry point gets control when a warm start occurs. A warm start is performed whenever a user program branches to location 0000H, or when the CPU is reset from the front panel. The CP/M system must be loaded from the first two tracks of drive A up to, but not including, the BIOS (or CBIOS, if you have completed your patch). System parameters must be initialized as shown below:

location 0,1,2 set to JMP WBOOT for warm starts
(0000H: JMP 4A03H+b)
location 3 set initial value of IOBYTE, if
implemented in your CBIOS
location 5,6,7 set to JMP BDOS, which is the
primary entry point to CP/M for
transient programs. (0005H: JMP
3C06H+b)

(see Section 9 for complete details of page zero use)
Upon completion of the initialization, the WBOOT program must branch to the CCP at 3400H+b to (re)start the system. Upon entry to the CCP, register C is set to the drive to select after system initialization.

CONST Sample the status of the currently assigned console device and return 0FFH in register A if a character is ready to read, and 00H in register A if no console characters are ready.

CONIN Read the next console character into register A, and

set the parity bit (high order bit) to zero. If no console character is ready, wait until a character is typed before returning.

- CONOUT Send the character from register C to the console output device. The character is in ASCII, with high order parity bit set to zero. You may want to include a time-out on a line feed or carriage return, if your console device requires some time interval at the end of the line (such as a TI Silent 700 terminal). You can, if you wish, filter out control characters which cause your console device to react in a strange way (a control-z causes the Lear Seigler terminal to clear the screen, for example).
- LIST Send the character from register C to the currently assigned listing device. The character is in ASCII with zero parity.
- PUNCH Send the character from register C to the currently assigned punch device. The character is in ASCII with zero parity.
- READER Read the next character from the currently assigned reader device into register A with zero parity (high order bit must be zero), an end of file condition is reported by returning an ASCII control-z (1AH).
- HOME Return the disk head of the currently selected disk (initially disk A) to the track 00 position. If your controller allows access to the track 0 flag from the drive, step the head until the track 0 flag is detected. If your controller does not support this feature, you can translate the HOME call into a call on SETTRK with a parameter of 0.
- SELDSK Select the disk drive given by register C for further operations, where register C contains 0 for drive A, 1 for drive B, and so-forth up to 15 for drive P (the standard CP/M distribution version supports four drives). On each disk select, SELDSK must return in HL the base address of a 16-byte area, called the Disk Parameter Header, described in the Section 10. For standard floppy disk drives, the contents of the header and associated tables does not change, and thus the program segment included in the sample CBIOS performs this operation automatically. If there is an attempt to select a non-existent drive, SELDSK returns HL=0000H as an error indicator. Although SELDSK must return the header address on each call, it is advisable to postpone the actual physical disk select operation until an I/O function (seek, read or write) is actually performed, since disk selects often occur without ultimately performing any disk I/O, and many controllers will unload the head of the current disk

before selecting the new drive. This would cause an excessive amount of noise and disk wear.

SETTRK

Register BC contains the track number for subsequent disk accesses on the currently selected drive. You can choose to seek the selected track at this time, or delay the seek until the next read or write actually occurs. Register BC can take on values in the range 0-76 corresponding to valid track numbers for standard floppy disk drives, and 0-65535 for non-standard disk subsystems.

SETSEC

Register BC contains the sector number (1 through 26) for subsequent disk accesses on the currently selected drive. You can choose to send this information to the controller at this point, or instead delay sector selection until a read or write operation occurs.

SETDMA

Register BC contains the DMA (disk memory access) address for subsequent read or write operations. For example, if B = 00H and C = 80H when SETDMA is called, then all subsequent read operations read their data into 80H through 0FFH, and all subsequent write operations get their data from 80H through 0FFH, until the next call to SETDMA occurs. The initial DMA address is assumed to be 80H. Note that the controller need not actually support direct memory access. If, for example, all data is received and sent through I/O ports, the CBIOS which you construct will use the 128 byte area starting at the selected DMA address for the memory buffer during the following read or write operations.

READ

Assuming the drive has been selected, the track has been set, the sector has been set, and the DMA address has been specified, the READ subroutine attempts to read one sector based upon these parameters, and returns the following error codes in register A:

- 0 no errors occurred
- 1 non-recoverable error condition occurred

Currently, CP/M responds only to a zero or non-zero value as the return code. That is, if the value in register A is 0 then CP/M assumes that the disk operation completed properly. If an error occurs, however, the CBIOS should attempt at least 10 retries to see if the error is recoverable. When an error is reported the BDOS will print the message "BDOS ERR ON x: BAD SECTOR". The operator then has the option of typing <cr> to ignore the error, or ctl-C to abort.

WRITE

Write the data from the currently selected DMA address to the currently selected drive, track, and sector. The data should be marked as "non deleted data" to

maintain compatibility with other CP/M systems. The error codes given in the READ command are returned in register A, with error recovery attempts as described above.

LISTST Return the ready status of the list device. Used by the DESPOOL program to improve console response during its operation. The value 00 is returned in A if the list device is not ready to accept a character, and 0FFH if a character can be sent to the printer. Note that a 00 value always suffices.

SECTTRAN Performs sector logical to physical sector translation in order to improve the overall response of CP/M. Standard CP/M systems are shipped with a "skew factor" of 6, where six physical sectors are skipped between each logical read operation. This skew factor allows enough time between sectors for most programs to load their buffers without missing the next sector. In particular computer systems which use fast processors, memory, and disk subsystems, the skew factor may be changed to improve overall response. Note, however, that you should maintain a single density IBM compatible version of CP/M for information transfer into and out of your computer system, using a skew factor of 6. In general, SECTTRAN receives a logical sector number in BC, and a translate table address in DE. The sector number is used as an index into the translate table, with the resulting physical sector number in HL. For standard systems, the tables and indexing code is provided in the CBIOS and need not be changed.

7. A SAMPLE BIOS

The program shown in Appendix C can serve as a basis for your first BIOS. The simplest functions are assumed in this BIOS, so that you can enter it through the front panel, if absolutely necessary. Note that the user must alter and insert code into the subroutines for CONST, CONIN, CONOUT, READ, WRITE, and WAITIO subroutines. Storage is reserved for user-supplied code in these regions. The scratch area reserved in page zero (see Section 9) for the BIOS is used in this program, so that it could be implemented in ROM, if desired.

Once operational, this skeletal version can be enhanced to print the initial sign-on message and perform better error recovery. The subroutines for LIST, PUNCH, and READER can be filled-out, and the IOBYTE function can be implemented.

8. A SAMPLE COLD START LOADER

The program shown in Appendix D can serve as a basis for your cold start loader. The disk read function must be supplied by the user, and the program must be loaded somehow starting at location 0000. Note that space is reserved for your patch so that the total amount of storage required for the cold start loader is 128 bytes. Eventually, you will probably want to get this loader onto the first disk sector (track 0, sector 1), and cause your controller to load it into memory automatically upon system start-up. Alternatively, you may wish to place the cold start loader into ROM, and place it above the CP/M system. In this case, it will be necessary to originate the program at a higher address, and key-in a jump instruction at system start-up which branches to the loader. Subsequent warm starts will not require this key-in operation, since the entry point 'WBOOT' gets control, thus bringing the system in from disk automatically. Note also that the skeletal cold start loader has minimal error recovery, which may be enhanced on later versions.

9. RESERVED LOCATIONS IN PAGE ZERO

Main memory page zero, between locations 000H and 0FFH, contains several segments of code and data which are used during CP/M processing. The code and data areas are given below for reference purposes.

Locations from to	Contents
0000H - 0002H	Contains a jump instruction to the warm start entry point at location 4A03H+b. This allows a simple programmed restart (JMP 0000H) or manual restart from the front panel.
0003H - 0003H	Contains the Intel standard IOBYTE, which is optionally included in the user's CBIOS, as described in Section 6.
0004H - 0004H	Current default drive number (0=A, ..., 15=P).
0005H - 0007H	Contains a jump instruction to the BDOS, and serves two purposes: JMP 0005H provides the primary entry point to the BDOS, as described in the manual "CP/M Interface Guide," and LHL 0006H brings the address field of the instruction to the HL register pair. This value is the lowest address in memory used by CP/M (assuming the CCP is being overlayed). Note that the DDT program will change the address field to reflect the reduced memory size in debug mode.
0008H - 0027H	(interrupt locations 1 through 5 not used)
0030H - 0037H	(interrupt location 6, not currently used - reserved)
0038H - 003AH	Restart 7 - Contains a jump instruction into the DDT or SID program when running in debug mode for programmed breakpoints, but is not otherwise used by CP/M.
003BH - 003FH	(not currently used - reserved)
0040H - 004FH	16 byte area reserved for scratch by CBIOS, but is not used for any purpose in the distribution version of CP/M
0050H - 005BH	(not currently used - reserved)
005CH - 007CH	default file control block produced for a transient program by the Console Command Processor.
007DH - 007FH	Optional default random record position

(All Information Contained Herein is Proprietary to Digital Research.)

0080H - 00FFH default 128 byte disk buffer (also filled with
the command line when a transient is loaded
under the CCP).

Note that this information is set-up for normal operation under the CP/M system, but can be overwritten by a transient program if the BDOS facilities are not required by the transient.

If, for example, a particular program performs only simple I/O and must begin execution at location 0, it can be first loaded into the TPA, using normal CP/M facilities, with a small memory move program which gets control when loaded (the memory move program must get control from location 0100H, which is the assumed beginning of all transient programs). The move program can then proceed to move the entire memory image down to location 0, and pass control to the starting address of the memory load. Note that if the BIOS is overwritten, or if location 0 (containing the warm start entry point) is overwritten, then the programmer must bring the CP/M system back into memory with a cold start sequence.

10. DISK PARAMETER TABLES.

Tables are included in the BIOS which describe the particular characteristics of the disk subsystem used with CP/M. These tables can be either hand-coded, as shown in the sample CBIOS in Appendix C, or automatically generated using the DISKDEF macro library, as shown in Appendix B. The purpose here is to describe the elements of these tables.

In general, each disk drive has an associated (16-byte) disk parameter header which both contains information about the disk drive and provides a scratchpad area for certain BDOS operations. The format of the disk parameter header for each drive is shown below

Disk			Parameter		Header				
	XLT	0000	0000	0000	DIRBUF	DPB	CSV	ALV	
	16b		16b		16b		16b		16b

where each element is a word (16-bit) value. The meaning of each Disk Parameter Header (DPH) element is

XLT	Address of the logical to physical translation vector, if used for this particular drive, or the value 0000H if no sector translation takes place (i.e, the physical and logical sector numbers are the same). Disk drives with identical sector skew factors share the same translate tables.
0000	Scratchpad values for use within the BDOS (initial value is unimportant).
DIRBUF	Address of a 128 byte scratchpad area for directory operations within BDOS. All DPH's address the same scratchpad area.
DPB	Address of a disk parameter block for this drive. Drives with identical disk characteristics address the same disk parameter block.
CSV	Address of a scratchpad area used for software check for changed disks. This address is different for each DPH.
ALV	Address of a scratchpad area used by the BDOS to keep disk storage allocation information. This address is different for each DPH.

Given n disk drives, the DPH's are arranged in a table whose first row of 16 bytes corresponds to drive 0, with the last row corresponding to drive n-1. The table thus appears as

DPBASE:

```

-----
00 |XLT 00| 0000 | 0000 | 0000 |DIRBUF|DBP 00|CSV 00|ALV 00|
-----
01 |XLT 01| 0000 | 0000 | 0000 |DIRBUF|DBP 01|CSV 01|ALV 01|
-----
                                (and so-forth through)
-----
n-1|XLTn-1| 0000 | 0000 | 0000 |DIRBUF|DBPn-1|CSVn-1|ALVn-1|
-----

```

where the label DPBASE defines the base address of the DPH table.

A responsibility of the SELDSK subroutine is to return the base address of the DPH for the selected drive. The following sequence of operations returns the table address, with a 0000H returned if the selected drive does not exist.

```

NDISKS   EQU      4   ;NUMBER OF DISK DRIVES
.....
SELDISK:
;SELECT DISK GIVEN BY BC
LXI      H,0000H   ;ERROR CODE
MOV      A,C       ;DRIVE OK?
CPI      NDISKS    ;CY IF SO
RNC      ;RET IF ERROR
;NO ERROR, CONTINUE
MOV      L,C       ;LOW(DISK)
MOV      H,B       ;HIGH(DISK)
DAD      H         ;*2
DAD      H         ;*4
DAD      H         ;*8
DAD      H         ;*16
LXI      D,DPBASE  ;FIRST DPH
DAD      D         ;DPH(DISK)
RET

```

The translation vectors (XLT 00 through XLTn-1) are located elsewhere in the BIOS, and simply correspond one-for-one with the logical sector numbers zero through the sector count-1. The Disk Parameter Block (DPB) for each drive is more complex. A particular DPB, which is addressed by one or more DPH's, takes the general form

```

-----
| SPT | BSH|BLM|EXM| DSM |  DRM |AL0|AL1| CKS |  OFF |
-----
16b   8b  8b  8b   16b   16b   8b  8b   16b   16b
-----

```

where each is a byte or word value, as shown by the "8b" or "16b" indicator below the field.

SPT is the total number of sectors per track

BSH is the data allocation block shift factor, determined by the data block allocation size.

(All Information Contained Herein is Proprietary to Digital Research.)

EXM is the extent mask, determined by the data block allocation size and the number of disk blocks.

DSM determines the total storage capacity of the disk drive

DRM determines the total number of directory entries which can be stored on this drive AL0,AL1 determine reserved directory blocks.

CKS is the size of the directory check vector

OFF is the number of reserved tracks at the beginning of the (logical) disk.

The values of BSH and BLM determine (implicitly) the data allocation size BLS, which is not an entry in the disk parameter block. Given that the designer has selected a value for BLS, the values of BSH and BLM are shown in the table below

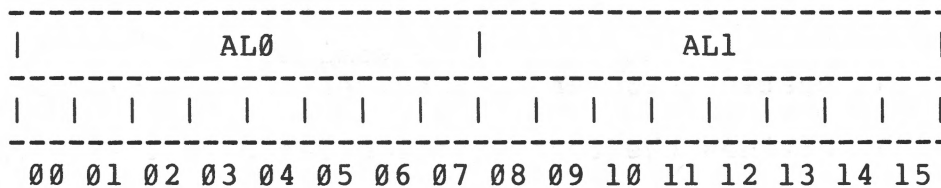
BLS	BSH	BLM
1,024	3	7
2,048	4	15
4,096	5	31
8,192	6	63
16,384	7	127

where all values are in decimal. The value of EXM depends upon both the BLS and whether the DSM value is less than 256 or greater than 255, as shown in the following table

BLS	DSM < 256	DSM > 255
1,024	0	N/A
2,048	1	0
4,096	3	1
8,192	7	3
16,384	15	7

The value of DSM is the maximum data block number supported by this particular drive, measured in BLS units. The product BLS times (DSM+1) is the total number of bytes held by the drive and, of course, must be within the capacity of the physical disk, not counting the reserved operating system tracks.

The DRM entry is the one less than the total number of directory entries, which can take on a 16-bit value. The values of AL0 and AL1, however, are determined by DRM. The two values AL0 and AL1 can together be considered a string of 16-bits, as shown below.



where position 00 corresponds to the high order bit of the byte labelled AL0, and 15 corresponds to the low order bit of the byte labelled AL1. Each bit position reserves a data block for number of directory entries, thus allowing a total of 16 data blocks to be assigned for directory entries (bits are assigned starting at 00 and filled to the right until position 15). Each directory entry occupies 32 bytes, resulting in the following table

BLS	Directory Entries
1,024	32 times # bits
2,048	64 times # bits
4,096	128 times # bits
8,192	256 times # bits
16,384	512 times # bits

Thus, if DRM = 127 (128 directory entries), and BLS = 1024, then there are 32 directory entries per block, requiring 4 reserved blocks. In this case, the 4 high order bits of AL0 are set, resulting in the values AL0 = 0F0H and AL1 = 00H.

The CKS value is determined as follows: if the disk drive media is removable, then $CKS = (DRM+1)/4$, where DRM is the last directory entry number. If the media is fixed, then set $CKS = 0$ (no directory records are checked in this case).

Finally, the OFF field determines the number of tracks which are skipped at the beginning of the physical disk. This value is automatically added whenever SETTRK is called, and can be used as a mechanism for skipping reserved operating system tracks, or for partitioning a large disk into smaller segmented sections.

To complete the discussion of the DPB, recall that several DPH's can address the same DPB if their drive characteristics are identical. Further, the DPB can be dynamically changed when a new drive is addressed by simply changing the pointer in the DPH since the BDOS copies the DPB values to a local area whenever the SELDSK function is invoked.

Returning back to the DPH for a particular drive, note that the two address values CSV and ALV remain. Both addresses reference an area of uninitialized memory following the BIOS. The areas must be unique for each drive, and the size of each area is determined by the values in the DPB.

The size of the area addressed by CSV is CKS bytes, which is sufficient to hold the directory check information for this particular drive. If $CKS = (DRM+1)/4$, then you must reserve $(DRM+1)/4$ bytes for directory check use. If $CKS = 0$, then no storage is reserved.

(All Information Contained Herein is Proprietary to Digital Research.)

The size of the area addressed by ALV is determined by the maximum number of data blocks allowed for this particular disk, and is computed as $(DSM/8)+1$.

The CBIOS shown in Appendix C demonstrates an instance of these tables for standard 8" single density drives. It may be useful to examine this program, and compare the tabular values with the definitions given above.

11. THE DISKDEF MACRO LIBRARY.

A macro library is shown in Appendix F, called DISKDEF, which greatly simplifies the table construction process. You must have access to the MAC macro assembler, of course, to use the DISKDEF facility, while the macro library is included with all CP/M 2.0 distribution disks.

A BIOS disk definition consists of the following sequence of macro statements:

```
MACLIB   DISKDEF
.....
DISKS     n
DISKDEF   0,...
DISKDEF   1,...
.....
DISKDEF   n-1
.....
ENDEF
```

where the MACLIB statement loads the DISKDEF.LIB file (on the same disk as your BIOS) into MAC's internal tables. The DISKS macro call follows, which specifies the number of drives to be configured with your system, where n is an integer in the range 1 to 16. A series of DISKDEF macro calls then follow which define the characteristics of each logical disk, 0 through n-1 (corresponding to logical drives A through P). Note that the DISKS and DISKDEF macros generate the in-line fixed data tables described in the previous section, and thus must be placed in a non-executable portion of your BIOS, typically directly following the BIOS jump vector.

The remaining portion of your BIOS is defined following the DISKDEF macros, with the ENDEF macro call immediately preceding the END statement. The ENDEF (End of Diskdef) macro generates the necessary uninitialized RAM areas which are located in memory above your BIOS.

The form of the DISKDEF macro call is

```
DISKDEF  dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
```

where

dn	is the logical disk number, 0 to n-1
fsc	is the first physical sector number (0 or 1)
lsc	is the last sector number
skf	is the optional sector skew factor
bls	is the data allocation block size
dir	is the number of directory entries
cks	is the number of "checked" directory entries
ofs	is the track offset to logical track 00
[0]	is an optional 1.4 compatibility flag

The value "dn" is the drive number being defined with this DISKDEF

(All Information Contained Herein is Proprietary to Digital Research.)

macro invocation. The "fsc" parameter accounts for differing sector numbering systems, and is usually 0 or 1. The "lsc" is the last numbered sector on a track. When present, the "skf" parameter defines the sector skew factor which is used to create a sector translation table according to the skew. If the number of sectors is less than 256, a single-byte table is created, otherwise each translation table element occupies two bytes. No translation table is created if the skf parameter is omitted (or equal to 0). The "bls" parameter specifies the number of bytes allocated to each data block, and takes on the values 1024, 2048, 4096, 8192, or 16384. Generally, performance increases with larger data block sizes since there are fewer directory references and logically connected data records are physically close on the disk. Further, each directory entry addresses more data and the BIOS-resident ram space is reduced. The "dks" specifies the total disk size in "bls" units. That is, if the bls = 2048 and dks = 1000, then the total disk capacity is 2,048,000 bytes. If dks is greater than 255, then the block size parameter bls must be greater than 1024. The value of "dir" is the total number of directory entries which may exceed 255, if desired. The "cks" parameter determines the number of directory items to check on each directory scan, and is used internally to detect changed disks during system operation, where an intervening cold or warm start has not occurred (when this situation is detected, CP/M automatically marks the disk read/only so that data is not subsequently destroyed). As stated in the previous section, the value of cks = dir when the media is easily changed, as is the case with a floppy disk subsystem. If the disk is permanently mounted, then the value of cks is typically 0, since the probability of changing disks without a restart is quite low. The "ofs" value determines the number of tracks to skip when this particular drive is addressed, which can be used to reserve additional operating system space or to simulate several logical drives on a single large capacity physical drive. Finally, the [0] parameter is included when file compatibility is required with versions of 1.4 which have been modified for higher density disks. This parameter ensures that only 16K is allocated for each directory record, as was the case for previous versions. Normally, this parameter is not included.

For convenience and economy of table space, the special form

DISKDEF i,j

gives disk i the same characteristics as a previously defined drive j. A standard four-drive single density system, which is compatible with version 1.4, is defined using the following macro invocations:

```

DISKS      4
DISKDEF    0,1,26,6,1024,243,64,64,2
DISKDEF    1,0
DISKDEF    2,0
DISKDEF    3,0
...
ENDEF

```

with all disks having the same parameter values of 26 sectors per track (numbered 1 through 26), with 6 sectors skipped between each access, 1024 bytes per data block, 243 data blocks for a total of 243k byte disk capacity, 64 checked directory entries, and two operating system tracks.

The DISKS macro generates n Disk Parameter Headers (DPH's), starting at the DPH table address DPBASE generated by the macro. Each disk header block contains sixteen bytes, as described above, and correspond one-for-one to each of the defined drives. In the four drive standard system, for example, the DISKS macro generates a table of the form:

```

DPBASE EQU $
DPE0:  DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV0,ALV0
DPE1:  DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV1,ALV1
DPE2:  DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV2,ALV2
DPE3:  DW  XLT0,0000H,0000H,0000H,DIRBUF,DPB0,CSV3,ALV3

```

where the DPH labels are included for reference purposes to show the beginning table addresses for each drive 0 through 3. The values contained within the disk parameter header are described in detail in the previous section. The check and allocation vector addresses are generated by the ENDEF macro in the ram area following the BIOS code and tables.

Note that if the "skf" (skew factor) parameter is omitted (or equal to 0), the translation table is omitted, and a 0000H value is inserted in the XLT position of the disk parameter header for the disk. In a subsequent call to perform the logical to physical translation, SECTRAN receives a translation table address of DE = 0000H, and simply returns the original logical sector from BC in the HL register pair. A translate table is constructed when the skf parameter is present, and the (non-zero) table address is placed into the corresponding DPH's. The table shown below, for example, is constructed when the standard skew factor skf = 6 is specified in the DISKDEF macro call:

```

XLT0:  DB  1,7,13,19,25,5,11,17,23,3,9,15,21
        DB  2,8,14,20,26,6,12,18,24,4,10,16,22

```

Following the ENDEF macro call, a number of uninitialized data areas are defined. These data areas need not be a part of the BIOS which is loaded upon cold start, but must be available between the BIOS and the end of memory. The size of the uninitialized RAM area is determined by EQU statements generated by the ENDEF macro. For a standard four-drive system, the ENDEF macro might produce

(All Information Contained Herein is Proprietary to Digital Research.)

```

4C72 =      BEGDAT EQU $
          (data areas)
4DB0 =      ENDDAT EQU $
013C =      DATSIZ EQU $-BEGDAT

```

which indicates that uninitialized RAM begins at location 4C72H, ends at 4DB0H-1, and occupies 013CH bytes. You must ensure that these addresses are free for use after the system is loaded.

After modification, you can use the STAT program to check your drive characteristics, since STAT uses the disk parameter block to decode the drive information. The STAT command form

STAT d:DSK:

decodes the disk parameter block for drive d (d=A,...,P) and displays the values shown below:

```

r: 128 Byte Record Capacity
k: Kilobyte Drive Capacity
d: 32 Byte Directory Entries
c: Checked Directory Entries
e: Records/ Extent
b: Records/ Block
s: Sectors/ Track
t: Reserved Tracks

```

Three examples of DISKDEF macro invocations are shown below with corresponding STAT parameter values (the last produces a full 8-megabyte system).

```

DISKDEF 0,1,58,,2048,256,128,128,2
r=4096, k=512, d=128, c=128, e=256, b=16, s=58, t=2

```

```

DISKDEF 0,1,58,,2048,1024,300,0,2
r=16384, k=2048, d=300, c=0, e=128, b=16, s=58, t=2

```

```

DISKDEF 0,1,58,,16384,512,128,128,2
r=65536, k=8192, d=128, c=128, e=1024, b=128, s=58, t=2

```

12. SECTOR BLOCKING AND DEBLOCKING.

Upon each call to the BIOS WRITE entry point, the CP/M BDOS includes information which allows effective sector blocking and deblocking where the host disk subsystem has a sector size which is a multiple of the basic 128-byte unit. The purpose here is to present a general-purpose algorithm which can be included within your BIOS which uses the BDOS information to perform the operations automatically.

Upon each call to WRITE, the BDOS provides the following information in register C:

0	=	normal sector write
1	=	write to directory sector
2	=	write to the first sector of a new data block

Condition 0 occurs whenever the next write operation is into a previously written area, such as a random mode record update, when the write is to other than the first sector of an unallocated block, or when the write is not into the directory area. Condition 1 occurs when a write into the directory area is performed. Condition 2 occurs when the first record (only) of a newly allocated data block is written. In most cases, application programs read or write multiple 128 byte sectors in sequence, and thus there is little overhead involved in either operation when blocking and deblocking records since pre-read operations can be avoided when writing records.

Appendix G lists the blocking and deblocking algorithms in skeletal form (this file is included on your CP/M disk). Generally, the algorithms map all CP/M sector read operations onto the host disk through an intermediate buffer which is the size of the host disk sector. Throughout the program, values and variables which relate to the CP/M sector involved in a seek operation are prefixed by "sek," while those related to the host disk system are prefixed by "hst." The equate statements beginning on line 29 of Appendix G define the mapping between CP/M and the host system, and must be changed if other than the sample host system is involved.

The entry points BOOT and WBOOT must contain the initialization code starting on line 57, while the SELDSK entry point must be augmented by the code starting on line 65. Note that although the SELDSK entry point computes and returns the Disk Parameter Header address, it does not physically select the host disk at this point (it is selected later at READHST or WRITEHST). Further, SETTRK, SETTRK, and SETDMA simply store the values, but do not take any other action at this point. SECTTRAN performs a trivial trivial function of returning the physical sector number.

The principal entry points are READ and WRITE, starting on lines 110 and 125, respectively. These subroutines take the place of your previous READ and WRITE operations.

The actual physical read or write takes place at either WRITEHST or READHST, where all values have been prepared: hstdsk is the host

(All Information Contained Herein is Proprietary to Digital Research.)

disk number, hsttrk is the host track number, and hstsec is the host sector number (which may require translation to a physical sector number). You must insert code at this point which performs the full host sector read or write into, or out of, the buffer at hstbuf of length hstsiz. All other mapping functions are performed by the algorithms.

This particular algorithm was tested using an 80 megabyte hard disk unit which was originally configured for 128 byte sectors, producing approximately 35 megabytes of formatted storage. When configured for 512 byte host sectors, usable storage increased to 57 megabytes, with a corresponding 400% improvement in overall response. In this situation, there is no apparent overhead involved in deblocking sectors, with the advantage that user programs still maintain the (less memory consuming) 128-byte sectors. This is primarily due, of course, to the information provided by the BDOS which eliminates the necessity for pre-read operations to take place.

APPENDIX A: THE MDS COLD START LOADER

```

;      MDS-800 Cold Start Loader for CP/M 2.0
;
;      Version 2.0 August, 1979
;
0000 = false equ 0
ffff = true equ not false
0000 = testing equ false
;
        if testing
bias equ 03400h
        endif
        if not testing
0000 = bias equ 0000h
        endif
0000 = cpmb equ bias ;base of dos load
0806 = bdos equ 806h+bias ;entry to dos for calls
1880 = bdose equ 1880h+bias ;end of dos load
1600 = boot equ 1600h+bias ;cold start entry point
1603 = rboot equ boot+3 ;warm start entry point
;
3000 org 3000h ;loaded here by hardware
;
1880 = bdosl equ bdose-cpmb
0002 = ntrks equ 2 ;tracks to read
0031 = bdoss equ bdosl/128 ;# sectors in bdos
0019 = bdos0 equ 25 ;# on track 0
0018 = bdosl equ bdoss-bdos0 ;# on track 1
;
f800 = mon80 equ 0f800h ;intel monitor base
ff0f = rmon80 equ 0ff0fh ;restart location for mon80
0078 = base equ 078h ;'base' used by controller
0079 = rtype equ base+1 ;result type
007b = rbyte equ base+3 ;result byte
007f = reset equ base+7 ;reset controller
;
0078 = dstat equ base ;disk status port
0079 = ilow equ base+1 ;low iopb address
007a = ihigh equ base+2 ;high iopb address
00ff = bsw equ 0ffh ;boot switch
0003 = recal equ 3h ;recalibrate selected drive
0004 = readf equ 4h ;disk read function
0100 = stack equ 100h ;use end of boot for stack
;
rstart:
3000 310001 lxi sp,stack;in case of call to mon80
; clear disk status
3003 db79 in rtype
3005 db7b in rbyte
; check if boot switch is off
coldstart:
3007 dbff in bsw
3009 e602 ani 02h ;switch on?
300b c20730 jnz coldstart

```

```

;      clear the controller
300e d37f      out      reset      ;logic cleared
;
;
3010 0602      mvi      b,ntrks ;number of tracks to read
3012 214230    lxi      h,iopb0
;
start:
;
;      read first/next track into cpmb
3015 7d        mov      a,l
3016 d379      out      ilow
3018 7c        mov      a,h
3019 d37a      out      ihigh
301b db78      wait0:   in      dstat
301d e604      ani      4
301f calb30    jz       wait0
;
;      check disk status
3022 db79      in      rtype
3024 e603      ani      11b
3026 fe02      cpi      2
;
;      if      testing
;      cnc      rmon80 ;go to monitor if 11 or 10
;      endif
;      if      not testing
3028 d20030    jnc      rstart ;retry the load
;      endif
;
;      in      rbyte ;i/o complete, check status
;      if not ready, then go to mon80
302b db7b      in      rbyte
302d 17        ral
302e dc0fff    cc      rmon80 ;not ready bit set
3031 1f        rar      ;restore
3032 e61e      ani      11110b ;overrun/addr err/seek/crc
;
;      if      testing
;      cnz      rmon80 ;go to monitor
;      endif
;      if      not testing
3034 c20030    jnz      rstart ;retry the load
;      endif
;
;
3037 110700    lxi      d,iopbl ;length of iopb
303a 19        dad      d ;addressing next iopb
303b 05        dcr      b ;count down tracks
303c c21530    jnz      start
;
;
;      jmp boot, print message, set-up jmps
303f c30016    jmp      boot
;
;      parameter blocks

```

```

3042 80      iopb0:  db      80h      ;iocw, no update
3043 04              db      readf    ;read function
3044 19              db      bdos0    ;# sectors to read trk 0
3045 00              db      0        ;track 0
3046 02              db      2        ;start with sector 2, trk 0
3047 0000      dw      cpmb         ;start at base of bdos
0007 =      iopb1  equ      $-iopb0
;
3049 80      iopb1:  db      80h
304a 04              db      readf
304b 18              db      bdosl    ;sectors to read on track 1
304c 01              db      1        ;track 1
304d 01              db      1        ;sector 1
304e 800c      dw      cpmb+bdos0*128 ;base of second rd
3050              end

```

APPENDIX B: THE MDS BASIC I/O SYSTEM (BIOS)

```

;      mds-800 i/o drivers for cp/m 2.0
;      (four drive single density version)
;
;      version 2.0 august, 1979
;
0014 = vers      equ      20      ;version 2.0
;
;      copyright (c) 1979
;      digital research
;      box 579, pacific grove
;      california, 93950
;
4a00      org      4a00h      ;base of bios in 20k system
3400 =    cpmb      equ      3400h      ;base of cpm ccp
3c06 =    bdos      equ      3c06h      ;base of bdos in 20k system
1600 =    cpml      equ      $-cpmb      ;length (in bytes) of cpm system
002c =    nsects    equ      cpml/128;number of sectors to load
0002 =    offset    equ      2          ;number of disk tracks used by cp
0004 =    cdisk     equ      0004h      ;address of last logged disk
0080 =    buff      equ      0080h      ;default buffer address
000a =    retry     equ      10         ;max retries on disk i/o before e
;
;      perform following functions
;      boot      cold start
;      wboot     warm start (save i/o byte)
;      (boot and wboot are the same for mds)
;      const     console status
;      reg-a = 00 if no character ready
;      reg-a = ff if character ready
;      conin     console character in (result in reg-a)
;      conout    console character out (char in reg-c)
;      list      list out (char in reg-c)
;      punch     punch out (char in reg-c)
;      reader    paper tape reader in (result to reg-a)
;      home      move to track 00
;
;      (the following calls set-up the io parameter bloc
;      mds, which is used to perform subsequent reads an
;      seldsk    select disk given by reg-c (0,1,2...)
;      settrk    set track address (0,...76) for sub r/w
;      setsec    set sector address (1,...,26)
;      setdma    set subsequent dma address (initially 80h)
;
;      read/write assume previous calls to set i/o parms
;      read      read track/sector to preset dma address
;      write     write track/sector from preset dma addres
;
;      jump vector for indiviual routines
4a00 c3b34a      jmp      boot
4a03 c3c34a wboote: jmp      wboot
4a06 c3614b      jmp      const
4a09 c3644b      jmp      conin
4a0c c36a4b      jmp      conout

```

```

4a0f c36d4b      jmp      list
4a12 c3724b      jmp      punch
4a15 c3754b      jmp      reader
4a18 c3784b      jmp      home
4alb c37d4b      jmp      seldsk
4ale c3a74b      jmp      settrk
4a21 c3ac4b      jmp      setsec
4a24 c3bb4b      jmp      setdma
4a27 c3cl4b      jmp      read
4a2a c3ca4b      jmp      write
4a2d c3704b      jmp      listst ;list status
4a30 c3bl4b      jmp      sectran

;
      maclib      diskdef ;load the disk definition library
disks      4      ;four disks
4a33+=      dpbase equ      $      ;base of disk parameter blocks
4a33+824a00 dpe0:  dw      xlt0,0000h ;translate table
4a37+000000 dw      0000h,0000h ;scratch area
4a3b+6e4c73 dw      dirbuf,dpb0 ;dir buff,param block
4a3f+0d4dee dw      csv0,alv0 ;check, alloc vectors
4a43+824a00 dpe1:  dw      xlt1,0000h ;translate table
4a47+000000 dw      0000h,0000h ;scratch area
4a4b+6e4c73 dw      dirbuf,dpb1 ;dir buff,param block
4a4f+3c4d1d dw      csv1,alv1 ;check, alloc vectors
4a53+824a00 dpe2:  dw      xlt2,0000h ;translate table
4a57+000000 dw      0000h,0000h ;scratch area
4a5b+6e4c73 dw      dirbuf,dpb2 ;dir buff,param block
4a5f+6b4d4c dw      csv2,alv2 ;check, alloc vectors
4a63+824a00 dpe3:  dw      xlt3,0000h ;translate table
4a67+000000 dw      0000h,0000h ;scratch area
4a6b+6e4c73 dw      dirbuf,dpb3 ;dir buff,param block
4a6f+9a4d7b dw      csv3,alv3 ;check, alloc vectors
      diskdef 0,1,26,6,1024,243,64,64,offset
4a73+=      dpb0   equ      $      ;disk parm block
4a73+1a00   dw      26      ;sec per track
4a75+03     db      3      ;block shift
4a76+07     db      7      ;block mask
4a77+00     db      0      ;extnt mask
4a78+f200   dw      242    ;disk size-1
4a7a+3f00   dw      63     ;directory max
4a7c+c0     db      192    ;alloc0
4a7d+00     db      0      ;alloc1
4a7e+1000   dw      16     ;check size
4a80+0200   dw      2      ;offset
4a82+=      xlt0   equ      $      ;translate table
4a82+01     db      1
4a83+07     db      7
4a84+0d     db      13
4a85+13     db      19
4a86+19     db      25
4a87+05     db      5
4a88+0b     db      11
4a89+11     db      17
4a8a+17     db      23
4a8b+03     db      3

```



```

;
; disk ports and commands
0078 = base equ 78h ;base of disk command io ports
0078 = dstat equ base ;disk status (input)
0079 = rtype equ base+1 ;result type (input)
007b = rbyte equ base+3 ;result byte (input)
;
0079 = ilow equ base+1 ;iopb low address (output)
007a = ihigh equ base+2 ;iopb high address (output)
;
0004 = readf equ 4h ;read function
0006 = writf equ 6h ;write function
0003 = recal equ 3h ;recalibrate drive
0004 = iordy equ 4h ;i/o finished mask
000d = cr equ 0dh ;carriage return
000a = lf equ 0ah ;line feed
;
signon: ;signon message: xxk cp/m vers y.y
4a9c 0d0a0a db cr,lf,lf
4a9f 3230 db '20' ;sample memory size
4aal 6b2043f db 'k cp/m vers '
4aad 322e30 db vers/10+'0','.',vers mod 10+'0'
4ab0 0d0a00 db cr,lf,0
;
boot: ;print signon message and go to ccp
; (note: mds boot initialized iobyte at 0003h)
4ab3 310001 lxi sp,buff+80h
4ab6 219c4a lxi h,signon
4ab9 cdd34b call prmsg ;print message
4abc af xra a ;clear accumulator
4abd 320400 sta cdisk ;set initially to disk a
4ac0 c30f4b jmp gccpm ;go to cp/m
;
;
wboot:; loader on track 0, sector 1, which will be skippe
; read cp/m from disk - assuming there is a 128 byt
; start.
;
4ac3 318000 lxi sp,buff ;using dma - thus 80 thru ff ok f
;
4ac6 0e0a mvi c,retry ;max retries
4ac8 c5 push b
wboot0: ;enter here on error retries
4ac9 010034 lxi b,cpmb ;set dma address to start of disk
4acc cdbb4b call setdma
4acf 0e00 mvi c,0 ;boot from drive 0
4ad1 cd7d4b call seldsk
4ad4 0e00 mvi c,0
4ad6 cda74b call settmk ;start with track 0
4ad9 0e02 mvi c,2 ;start reading sector 2
4adb cdac4b call setsec
;
; read sectors, count nsects to zero
4ade cl pop b ;10-error count
4adf 062c mvi b,nsects

```

```

rdsec: ;read next sector
4ael c5      push    b      ;save sector count
4ae2 cdcl4b  call    read
4ae5 c2494b  jnz     booterr ;retry if errors occur
4ae8 2a6c4c  lhld    iod     ;increment dma address
4aeb 118000  lxi     d,128    ;sector size
4aee 19      dad     d      ;incremented dma address in hl
4aef 44      mov     b,h
4af0 4d      mov     c,l    ;ready for call to set dma
4af1 cdbb4b  call    setdma
4af4 3a6b4c  lda     ios     ;sector number just read
4af7 fela    cpi     26     ;read last sector?
4af9 da054b  jc      rdl
;          must be sector 26, zero and go to next track
4afc 3a6a4c  lda     iot     ;get track to register a
4aff 3c      inr     a
4b00 4f      mov     c,a    ;ready for call
4b01 cda74b  call    settrk
4b04 af      xra     a      ;clear sector number
4b05 3c      rdl:    inr     a ;to next sector
4b06 4f      mov     c,a    ;ready for call
4b07 cdac4b  call    setsec
4b0a cl      pop     b      ;recall sector count
4b0b 05      dcr     b      ;done?
4b0c c2e14a  jnz     rdsec
;
;          done with the load, reset default buffer address
gocpm: ;(enter here from cold start boot)
;          enable rst0 and rst7
4b0f f3      di
4b10 3e12    mvi     a,12h   ;initialize command
4b12 d3fd    out     revrt
4b14 af      xra     a
4b15 d3fc    out     intc   ;cleared
4b17 3e7e    mvi     a,inte  ;rst0 and rst7 bits on
4b19 d3fc    out     intc
4b1b af      xra     a
4b1c d3f3    out     icon   ;interrupt control
;
;          set default buffer address to 80h
4ble 018000  lxi     b,buff
4b21 cdbb4b  call    setdma
;
;          reset monitor entry points
4b24 3ec3    mvi     a,jmp
4b26 320000  sta     0
4b29 21034a  lxi     h,wboote
4b2c 220100  shld    1      ;jmp wboot at location 00
4b2f 320500  sta     5
4b32 21063c  lxi     h,bdos
4b35 220600  shld    6      ;jmp bdos at location 5
4b38 323800  sta     7*8   ;jmp to mon80 (may have been chan
4b3b 2100f8  lxi     h,mon80
4b3e 223900  shld    7*8+1
;          leave iobyte set

```

```

;           previously selected disk was b, send parameter to
4b41 3a0400    lda      cdisk    ;last logged disk number
4b44 4f        mov      c,a      ;send to ccp to log it in
4b45 fb        ei
4b46 c30034    jmp      cpmb

;
;           error condition occurred, print message and retry
booterr:
4b49 cl        pop      b          ;recall counts
4b4a 0d        dcr      c
4b4b ca524b    jz       booter0
;           try again
4b4e c5        push     b
4b4f c3c94a    jmp      wboot0

;
booter0:
;           otherwise too many retries
4b52 215b4b    lxi      h,bootmsg
4b55 cdd34b    call     prmsg
4b58 c30fff    jmp      rmon80    ;mds hardware monitor

;
bootmsg:
4b5b 3f626f4   db       '?boot',0
;
;
const: ;console status to reg-a
;       (exactly the same as mds call)
4b61 c312f8    jmp      csts
;
conin: ;console character to reg-a
4b64 cd03f8    call     ci
4b67 e67f      ani      7fh      ;remove parity bit
4b69 c9        ret

;
conout: ;console character from c to console out
4b6a c309f8    jmp      co
;
list: ;list device out
;       (exactly the same as mds call)
4b6d c30ff8    jmp      lo
;
listst:
;       ;return list status
4b70 af        xra      a
4b71 c9        ret            ;always not ready

;
punch: ;punch device out
;       (exactly the same as mds call)
4b72 c30cf8    jmp      po
;
reader: ;reader character in to reg-a
;       (exactly the same as mds call)
4b75 c306f8    jmp      ri
;
home: ;move to home position

```

```

;      treat as track 00 seek
4b78 0e00      mvi      c,0
4b7a c3a74b    jmp      settrk
;
seldsk: ;select disk given by register c
4b7d 210000    lxi      h,0000h ;return 0000 if error
4b80 79        mov      a,c
4b81 fe04      cpi      ndisks ;too large?
4b83 d0        rnc      ;leave hl = 0000
;
4b84 e602      ani      10b      ;00 00 for drive 0,1 and 10 10 fo
4b86 32664c    sta      dbank    ;to select drive bank
4b89 79        mov      a,c      ;00, 01, 10, 11
4b8a e601      ani      lb      ;mds has 0,1 at 78, 2,3 at 88
4b8c b7        ora      a        ;result 00?
4b8d ca924b    jz       setdrive
4b90 3e30      mvi      a,00110000b ;selects drive 1 in bank
setdrive:
4b92 47        mov      b,a      ;save the function
4b93 21684c    lxi      h,iof    ;io function
4b96 7e        mov      a,m
4b97 e6cf      ani      11001111b ;mask out disk number
4b99 b0        ora      b        ;mask in new disk number
4b9a 77        mov      m,a      ;save it in iopb
4b9b 69        mov      l,c
4b9c 2600      mvi      h,0      ;hl=disk number
4b9e 29        dad      h        ;*2
4b9f 29        dad      h        ;*4
4ba0 29        dad      h        ;*8
4ba1 29        dad      h        ;*16
4ba2 11334a    lxi      d,dpbase
4ba5 19        dad      d        ;hl=disk header table address
4ba6 c9        ret
;
;
settrk: ;set track address given by c
4ba7 216a4c    lxi      h,iot
4baa 71        mov      m,c
4bab c9        ret
;
setsec: ;set sector number given by c
4bac 216b4c    lxi      h,ios
4baf 71        mov      m,c
4bb0 c9        ret
sectran:
;translate sector bc using table at de
4bb1 0600      mvi      b,0      ;double precision sector number i
4bb3 eb        xchg      ;translate table address to hl
4bb4 09        dad      b        ;translate(sector) address
4bb5 7e        mov      a,m      ;translated sector number to a
4bb6 326b4c    sta      ios
4bb9 6f        mov      l,a      ;return sector number in l
4bba c9        ret
;
setdma: ;set dma address given by regs b,c

```



```

4bbb 69          mov     l,c
4bbc 60          mov     h,b
4bbd 226c4c      shld    iod
4bc0 c9          ret

;
read:            ;read next disk record (assuming disk/trk/sec/dma
4bcl 0e04        mvi     c,readf ;set to read function
4bc3 cde04b      call    setfunc
4bc6 cdf04b      call    waitio ;perform read function
4bc9 c9          ret           ;may have error set in reg-a

;
;
write:           ;disk write function
4bca 0e06        mvi     c,writf
4bcc cde04b      call    setfunc ;set to write function
4bcf cdf04b      call    waitio
4bd2 c9          ret           ;may have error set

;
;
;               utility subroutines
prmsg:           ;print message at h,l to 0
4bd3 7e          mov     a,m
4bd4 b7          ora     a        ;zero?
4bd5 c8          rz

;               more to print
4bd6 e5          push    h
4bd7 4f          mov     c,a
4bd8 cd6a4b      call    conout
4bdb e1          pop     h
4bdc 23          inx     h
4bdd c3d34b      jmp     prmsg

;
;               setfunc:
;               set function for next i/o (command in reg-c)
4be0 21684c      lxi     h,iof    ;io function address
4be3 7e          mov     a,m      ;get it to accumulator for maskin
4be4 e6f8        ani     11111000b ;remove previous command
4be6 b1          ora     c        ;set to new command
4be7 77          mov     m,a      ;replaced in iopb

;               the mds-800 controller req's disk bank bit in sec
;               mask the bit from the current i/o function
4be8 e620        ani     00100000b ;mask the disk select bit
4bea 216b4c      lxi     h,ios     ;address the sector selec
4bed b6          ora     m        ;select proper disk bank
4bee 77          mov     m,a      ;set disk select bit on/o
4bef c9          ret

;
waitio:          mvi     c,retry ;max retries before perm error
rewait:          ;
;               start the i/o function and wait for completion
4bf2 cd3f4c      call    intype ;in rtype
4bf5 cd4c4c      call    inbyte ;clears the controller

;
4bf8 3a664c      lda     dbank    ;set bank flags

```

```

4bfb b7          ora      a          ;zero if drive 0,1 and nz
4bfc 3e67        mvi      a,iopb and 0ffh ;low address for iopb
4bfe 064c        mvi      b,iopb shr 8   ;high address for iopb
4c00 c20b4c      jnz      iodrl ;drive bank 1?
4c03 d379        out      ilow          ;low address to controlle
4c05 78          mov      a,b
4c06 d37a        out      ihigh ;high address
4c08 c3104c      jmp      wait0 ;to wait for complete

;
iodrl: ;drive bank 1
4c0b d389        out      ilow+10h      ;88 for drive bank 10
4c0d 78          mov      a,b
4c0e d38a        out      ihigh+10h

;
4c10 cd594c wait0: call      instat      ;wait for completion
4c13 e604        ani      iordy        ;ready?
4c15 cal04c      jz       wait0

;
; check io completion ok
4c18 cd3f4c      call      intype      ;must be io complete (00)
; 00 unlinked i/o complete, 01 linked i/o comple
; 10 disk status changed 11 (not used)
4c1b fe02        cpi      l0b         ;ready status change?
4c1d ca324c      jz       wready

;
; must be 00 in the accumulator
4c20 b7          ora      a
4c21 c2384c      jnz      werror      ;some other condition, re

;
; check i/o error bits
4c24 cd4c4c      call      inbyte
4c27 17          ral
4c28 da324c      jc       wready      ;unit not ready
4c2b 1f          rar
4c2c e6fe        ani      11111110b    ;any other errors?
4c2e c2384c      jnz      werror

;
; read or write is ok, accumulator contains zero
4c31 c9          ret

;
wready: ;not ready, treat as error for now
4c32 cd4c4c      call      inbyte      ;clear result byte
4c35 c3384c      jmp      trycount

;
werror: ;return hardware malfunction (crc, track, seek, e
; the mds controller has returned a bit in each pos
; of the accumulator, corresponding to the conditio
; 0 - deleted data (accepted as ok above)
; 1 - crc error
; 2 - seek error
; 3 - address error (hardware malfunction)
; 4 - data over/under flow (hardware malfunct
; 5 - write protect (treated as not ready)
; 6 - write error (hardware malfunction)
; 7 - not ready

```

```

;      (accumulator bits are numbered 7 6 5 4 3 2 1 0)
;
;      it may be useful to filter out the various condit
;      but we will get a permanent error message if it i
;      recoverable. in any case, the not ready conditio
;      treated as a separate condition for later improve
trycount:
;      register c contains retry count, decrement 'til z
4c38 0d      dcr      c
4c39 c2f24b   jnz      rewait ;for another try
;
;      cannot recover from error
4c3c 3e01     mvi      a,1      ;error code
4c3e c9       ret
;
;      intype, inbyte, instat read drive bank 00 or 10
4c3f 3a664c  intype: lda      dbank
4c42 b7       ora      a
4c43 c2494c   jnz      intypl ;skip to bank 10
4c46 db79     in       rtype
4c48 c9       ret
4c49 db89     intypl: in       rtype+10h      ;78 for 0,1 88 for 2,3
4c4b c9       ret
;
4c4c 3a664c  inbyte: lda      dbank
4c4f b7       ora      a
4c50 c2564c   jnz      inbytl
4c53 db7b     in       rbyte
4c55 c9       ret
4c56 db8b     inbytl: in       rbyte+10h
4c58 c9       ret
;
4c59 3a664c  instat: lda      dbank
4c5c b7       ora      a
4c5d c2634c   jnz      instal
4c60 db78     in       dstat
4c62 c9       ret
4c63 db88     instal: in       dstat+10h
4c65 c9       ret
;
;
;
;      data areas (must be in ram)
4c66 00      dbank: db      0      ;disk bank 00 if drive 0,1
;                      ;      10 if drive 2,3
;
;      iopb: ;io parameter block
4c67 80      db      80h      ;normal i/o operation
4c68 04      ioof: db      readf ;io function, initial read
4c69 01      ion:  db      1      ;number of sectors to read
4c6a 02      iot:  db      offset ;track number
4c6b 01      ios:  db      1      ;sector number
4c6c 8000     iod:  dw      buff   ;io address
;
;
;      define ram areas for bdos operation

```

```

                                endef
4c6e+=      begdat equ $
4c6e+      dirbuf: ds 128      ;directory access buffer
4ceet+      alv0:   ds 31
4d0d+      csv0:   ds 16
4d1d+      alv1:   ds 31
4d3c+      csv1:   ds 16
4d4c+      alv2:   ds 31
4d6b+      csv2:   ds 16
4d7b+      alv3:   ds 31
4d9a+      csv3:   ds 16
4daa+=      enddat equ $
013c+=      datsiz equ $-begdat
4daa      end

```

APPENDIX C: A SKELETAL CBIOS

```

; skeletal cbios for first level of cp/m 2.0 altera
;
0014 = msize equ 20 ;cp/m version memory size in kilo
;
; "bias" is address offset from 3400h for memory sy
; than 16k (referred to as "b" throughout the text)
;
0000 = bias equ (msize-20)*1024
3400 = ccp equ 3400h+bias ;base of ccp
3c06 = bdos equ ccp+806h ;base of bdos
4a00 = bios equ ccp+1600h ;base of bios
0004 = cdisk equ 0004h ;current disk number 0=a,...,15=p
0003 = iobyte equ 0003h ;intel i/o byte
;
4a00 org bios ;origin of this program
002c = nsects equ ($-ccp)/128 ;warm start sector count
;
; jump vector for individual subroutines
4a00 c39c4a jmp boot ;cold start
4a03 c3a64a wboote: jmp wboot ;warm start
4a06 c3114b jmp const ;console status
4a09 c3244b jmp conin ;console character in
4a0c c3374b jmp conout ;console character out
4a0f c3494b jmp list ;list character out
4a12 c34d4b jmp punch ;punch character out
4a15 c34f4b jmp reader ;reader character out
4a18 c3544b jmp home ;move head to home positi
4a1b c35a4b jmp seldsk ;select disk
4a1e c37d4b jmp settrk ;set track number
4a21 c3924b jmp setsec ;set sector number
4a24 c3ad4b jmp setdma ;set dma address
4a27 c3c34b jmp read ;read disk
4a2a c3d64b jmp write ;write disk
4a2d c34b4b jmp listst ;return list status
4a30 c3a74b jmp sectran ;sector translate
;
; fixed data tables for four-drive standard
;
; ibm-compatible 8" disks
;
; disk parameter header for disk 00
4a33 734a00 dpbase: dw trans,0000h
4a37 000000 dw 0000h,0000h
4a3b f04c8d dw dirbf,dpblk
4a3f ec4d70 dw chk00,all00
;
; disk parameter header for disk 01
4a43 734a00 dw trans,0000h
4a47 000000 dw 0000h,0000h
4a4b f04c8d dw dirbf,dpblk
4a4f fc4d8f dw chk01,all01
;
; disk parameter header for disk 02
4a53 734a00 dw trans,0000h
4a57 000000 dw 0000h,0000h
4a5b f04c8d dw dirbf,dpblk
4a5f 0c4eae dw chk02,all02

```



```

; disk parameter header for disk 03
4a63 734a00 dw trans,0000h
4a67 000000 dw 0000h,0000h
4a6b f04c8d dw dirbf,dpblk
4a6f 1c4ecd dw chk03,all03

;
; sector translate vector
trans: db 1,7,13,19 ;sectors 1,2,3,4
db 25,5,11,17 ;sectors 5,6,7,8
db 23,3,9,15 ;sectors 9,10,11,12
db 21,2,8,14 ;sectors 13,14,15,16
db 20,26,6,12 ;sectors 17,18,19,20
db 18,24,4,10 ;sectors 21,22,23,24
db 16,22 ;sectors 25,26

;
dpblk: ;disk parameter block, common to all disks
4a8d 1a00 dw 26 ;sectors per track
4a8f 03 db 3 ;block shift factor
4a90 07 db 7 ;block mask
4a91 00 db 0 ;null mask
4a92 f200 dw 242 ;disk size-1
4a94 3f00 dw 63 ;directory max
4a96 c0 db 192 ;alloc 0
4a97 00 db 0 ;alloc 1
4a98 1000 dw 16 ;check size
4a9a 0200 dw 2 ;track offset

;
; end of fixed tables

;
; individual subroutines to perform each function
boot: ;simplest case is to just perform parameter initi
4a9c af xra a ;zero in the accum
4a9d 320300 sta iobyte ;clear the iobyte
4aa0 320400 sta cdisk ;select disk zero
4aa3 c3ef4a jmp gocpm ;initialize and go to cp/

;
wboot: ;simplest case is to read the disk until all sect
4aa6 318000 lxi sp,80h ;use space below buffer f
4aa9 0e00 mvi c,0 ;select disk 0
4aab cd5a4b call seldsk
4aae cd544b call home ;go to track 00

;
4ab1 062c mvi b,nsects ;b counts # of sectors to
4ab3 0e00 mvi c,0 ;c has the current track
4ab5 1602 mvi d,2 ;d has the next sector to

; note that we begin by reading track 0, sector 2 s
; contains the cold start loader, which is skipped
4ab7 210034 lxi h,ccp ;base of cp/m (initial lo

load1: ;load one more sector
4aba c5 push b ;save sector count, current track
4abb d5 push d ;save next sector to read
4abc e5 push h ;save dma address
4abd 4a mov c,d ;get sector address to register c
4abe cd924b call setsec ;set sector address from register
4acl c1 pop b ;recall dma address to b,c

```

```

4ac2 c5      push    b      ;replace on stack for later recal
4ac3 cdad4b  call    setdma  ;set dma address from b,c
;
; drive set to 0, track set, sector set, dma address
4ac6 cdc34b  call    read
4ac9 fe00    cpi      00h    ;any errors?
4acb c2a64a  jnz      wboot    ;retry the entire boot if an erro
;
; no error, move to next sector
4ace e1      pop      h      ;recall dma address
4acf 118000  lxi      d,128    ;dma=dma+128
4ad2 19      dad      d      ;new dma address is in h,1
4ad3 d1      pop      d      ;recall sector address
4ad4 c1      pop      b      ;recall number of sectors remaini
4ad5 05      dcr      b      ;sectors=sectors-1
4ad6 caef4a  jz       gocpm    ;transfer to cp/m if all have bee
;
; more sectors remain to load, check for track chan
4ad9 14      inr      d
4ada 7a      mov      a,d    ;sector=27?, if so, change tracks
4adb felb    cpi      27
4add daba4a  jc       loadl   ;carry generated if sector<27
;
; end of current track, go to next track
4ae0 1601    mvi      d,1    ;begin with first sector of next
4ae2 0c      inr      c      ;track=track+1
;
; save register state, and change tracks
4ae3 c5      push    b
4ae4 d5      push    d
4ae5 e5      push    h
4ae6 cd7d4b  call    settrk  ;track address set from register
4ae9 e1      pop     h
4aea d1      pop     d
4aeb c1      pop     b
4aec c3ba4a  jmp     loadl   ;for another sector
;
; end of load operation, set parameters and go to c
gocpm:
4aef 3ec3    mvi      a,0c3h  ;c3 is a jmp instruction
4af1 320000  sta      0      ;for jmp to wboot
4af4 21034a  lxi      h,wboote  ;wboot entry point
4af7 220100  shld    1      ;set address field for jmp at 0
;
4afa 320500  sta      5      ;for jmp to bdos
4afd 21063c  lxi      h,bdos    ;bdos entry point
4b00 220600  shld    6      ;address field of jump at 5 to bd
;
4b03 018000  lxi      b,80h    ;default dma address is 80h
4b06 cdad4b  call    setdma
;
4b09 fb      ei       ;enable the interrupt system
4b0a 3a0400  lda      cdisk    ;get current disk number
4b0d 4f      mov      c,a    ;send to the ccp
4b0e c30034  jmp     ccp      ;go to cp/m for further processin

```

```

;
;
;       simple i/o handlers (must be filled in by user)
;       in each case, the entry point is provided, with s
;       to insert your own code
;
const:  ;console status, return 0ffh if character ready,
4b11    ds      10h      ;space for status subroutine
4b21 3e00 mvi      a,00h
4b23 c9    ret

;
conin:  ;console character into register a
4b24    ds      10h      ;space for input routine
4b34 e67f ani      7fh      ;strip parity bit
4b36 c9    ret

;
conout: ;console character output from register c
4b37 79    mov     a,c      ;get to accumulator
4b38    ds      10h      ;space for output routine
4b48 c9    ret

;
list:   ;list character from register c
4b49 79    mov     a,c      ;character to register a
4b4a c9    ret              ;null subroutine

;
listst: ;return list status (0 if not ready, 1 if ready)
4b4b af    xra      a      ;0 is always ok to return
4b4c c9    ret

;
punch:  ;punch character from register c
4b4d 79    mov     a,c      ;character to register a
4b4e c9    ret              ;null subroutine

;
;
reader: ;read character into register a from reader device
4b4f 3ela  mvi      a,lah    ;enter end of file for now (repla
4b51 e67f ani      7fh      ;remember to strip parity bit
4b53 c9    ret

;
;
;       i/o drivers for the disk follow
;       for now, we will simply store the parameters away
;       in the read and write subroutines
;
home:   ;move to the track 00 position of current drive
;       translate this call into a settrk call with param
4b54 0e00 mvi      c,0      ;select track 0
4b56 cd7d4b call     settrk
4b59 c9    ret              ;we will move to 00 on first read

;
seldsk: ;select disk given by register c
4b5a 210000 lxi      h,0000h ;error return code
4b5d 79    mov     a,c
4b5e 32ef4c sta     diskno
4b61 fe04  cpi      4      ;must be between 0 and 3

```

```

4b63 d0      rnc          ;no carry if 4,5,...
;           disk number is in the proper range
4b64          ds          10h      ;space for disk select
;           compute proper disk parameter header address
4b6e 3aef4c   lda          diskno
4b71 6f       mov          l,a      ;l=disk number 0,1,2,3
4b72 2600     mvi          h,0      ;high order zero
4b74 29       dad          h        ;*2
4b75 29       dad          h        ;*4
4b76 29       dad          h        ;*8
4b77 29       dad          h        ;*16 (size of each header)
4b78 11334a   lxi          d,dpbase
4b7b 19       dad          d        ;hl=.dpbase(diskno*16)
4b7c c9       ret

;
settrk: ;set track given by register c
4b7d 79       mov          a,c
4b7e 32e94c   sta          track
4b81          ds          10h      ;space for track select
4b91 c9       ret

;
setsec: ;set sector given by register c
4b92 79       mov          a,c
4b93 32eb4c   sta          sector
4b96          ds          10h      ;space for sector select
4ba6 c9       ret

;
sectran:
;           ;translate the sector given by bc using the
;           ;translate table given by de
4ba7 eb       xchg          ;hl=.trans
4ba8 09       dad          b        ;hl=.trans(sector)
4ba9 6e       mov          l,m      ;l = trans(sector)
4baa 2600     mvi          h,0      ;hl= trans(sector)
4bac c9       ret                ;with value in hl

;
setdma: ;set dma address given by registers b and c
4bad 69       mov          l,c      ;low order address
4bae 60       mov          h,b      ;high order address
4baf 22ed4c   shld         dmaad    ;save the address
4bb2          ds          10h      ;space for setting the dma address
4bc2 c9       ret

;
read: ;perform read operation (usually this is similar
;           so we will allow space to set up read command, th
;           common code in write)
4bc3          ds          10h      ;set up read command
4bd3 c3e64b   jmp          waitio   ;to perform the actual i/o

;
write: ;perform a write operation
4bd6          ds          10h      ;set up write command

;
waitio: ;enter here from read and write to perform the ac
;           operation. return a 00h in register a if the ope
;           properly, and 01h if an error occurs during the r

```

```

;
;       in this case, we have saved the disk number in 'd
;       the track number in 'track' (0-76
;       the sector number in 'sector' (1-
;       the dma address in 'dmaad' (0-655
4be6          ds      256      ;space reserved for i/o drivers
4ce6 3e01      mvi      a,1      ;error condition
4ce8 c9        ret              ;replaced when filled-in
;
;       the remainder of the cbios is reserved uninitiali
;       data area, and does not need to be a part of the
;       system memory image (the space must be available,
;       however, between "begdat" and "enddat").
;
4ce9 track: ds      2      ;two bytes for expansion
4ceb sector: ds     2      ;two bytes for expansion
4ced dmaad: ds      2      ;direct memory address
4cef diskno: ds     1      ;disk number 0-15
;
;       scratch ram area for bdos use
4cf0 = begdat equ $      ;beginning of data area
4cf0 dirbf: ds     128     ;scratch directory area
4d70 all00: ds     31      ;allocation vector 0
4d8f all01: ds     31      ;allocation vector 1
4dae all02: ds     31      ;allocation vector 2
4dcd all03: ds     31      ;allocation vector 3
4dec chk00: ds     16      ;check vector 0
4dfc chk01: ds     16      ;check vector 1
4e0c chk02: ds     16      ;check vector 2
4elc chk03: ds     16      ;check vector 3
;
4e2c = enddat equ $      ;end of data area
013c = datsiz equ $-begdat;size of data area
4e2c end

```


APPENDIX D: A SKELETAL GETSYS/PUTSYS PROGRAM

```

;      combined getsys and putsys programs from Sec 4.
;      Start the programs at the base of the TPA

0100          org      0100h

0014 =      msize     equ      20              ; size of cp/m in Kbytes

; "bias" is the amount to add to addresses for > 20k
;      (referred to as "b" throughout the text)

0000 =      bias      equ      (msize-20)*1024
3400 =      ccp        equ      3400h+bias
3c00 =      bdos       equ      ccp+0800h
4a00 =      bios       equ      ccp+1600h

;      getsys programs tracks 0 and 1 to memory at
;      3880h + bias

;      register          usage
;      a                  (scratch register)
;      b                  track count (0...76)
;      c                  sector count (1...26)
;      d,e                (scratch register pair)
;      h,l                load address
;      sp                 set to stack address

gstart:      ; start of getsys
0100 318033   lxi        sp,ccp-0080h      ; convenient plac
0103 218033   lxi        h,ccp-0080h      ; set initial loa
0106 0600     mvi        b,0              ; start with trac
rd$trk:      ; read next track
0108 0e01     mvi        c,1              ; each track star
rd$sec:      ;
010a cd0003   call       read$sec          ; get the next se
010d 118000   lxi        d,128            ; offset by one s
0110 19       dad        d                ; (hl=hl+128)
0111 0c       inr        c                ; next sector
0112 79       mov        a,c              ; fetch sector nu
0113 felb     cpi        27               ; and see if la
0115 da0a01   jc         rdsec             ; <, do one more

; arrive here at end of track, move to next track

0118 04       inr        b                ; track = track+1
0119 78       mov        a,b              ; check for last
011a fe02     cpi        2                ; track = 2 ?
011c da0801   jc         rd$trk           ; <, do another

; arrive here at end of load, halt for lack of anything b

011f fb       ei
0120 76       hlt

```

```

;      putsys program, places memory image starting at
;      3880h + bias back to tracks 0 and 1
;      start this program at the next page boundary

0200          org      ($+0100h) and 0ff00h

      putsys:
0200 318033    lxi      sp,ccp-0080h          ; convenient plac
0203 218033    lxi      h,ccp-0080h          ; start of dump
0206 0600      mvi      b,0                  ; start with trac

      wr$trk:
0208 0e01      mvi      c,1                  ; start with sect

      wr$sec:
020a cd0004    call     write$sec            ; write one secto
020d 118000    lxi      d,128                ; length of each
0210 19        dad      d                    ; <hl>=<hl> + 128
0211 0c        inr      c                    ; <c> = <c> + 1
0212 79        mov      a,c                  ; see if
0213 felb      cpi      27                   ; past end of t
0215 da0a02    jc       wr$sec               ; no, do another

; arrive here at end of track, move to next track

0218 04        inr      b                    ; track = track+1
0219 78        mov      a,b                  ; see if
021a fe02      cpi      2                    ; last track
021c da0802    jc       wr$trk              ; no, do another

; done with putsys, halt for lack of anything bette

021f fb        ei
0220 76        hlt

; user supplied subroutines for sector read and write

; move to next page boundary

0300          org      ($+0100h) and 0ff00h

      read$sec:
; read the next sector
; track in <b>,
; sector in <c>
; dmaaddr in <hl>

0300 c5        push     b
0301 e5        push     h

; user defined read operation goes here

0302          ds        64

0342 e1        pop      h
0343 c1        pop      b

```

```

0344 c9          ret
0400          org      ($+0100h) and 0ff00h      ; another page bo
write$sec:
                ; same parameters as read$sec

0400 c5          push   b
0401 e5          push   h

                ; user defined write operation goes here
0402          ds       64

0442 e1          pop    h
0443 c1          pop    b
0444 c9          ret

                ; end of getsys/putsys program

0445          end

```

APPENDIX E: A SKELETAL COLD START LOADER

; this is a sample cold start loader which, when modified
; resides on track 00, sector 01 (the first sector on the
; diskette). we assume that the controller has loaded
; this sector into memory upon system start-up (this pro-
; gram can be keyed-in, or can exist in read/only memory
; beyond the address space of the cp/m version you are
; running). the cold start loader brings the cp/m system
; into memory at "loadp" (3400h + "bias"). in a 20k
; memory system, the value of "bias" is 0000h, with large
; values for increased memory sizes (see section 2). afte
; loading the cp/m system, the clod start loader branches
; to the "boot" entry point of the bios, which begins at
; "bios" + "bias." the cold start loader is not used un-
; til the system is powered up again, as long as the bios
; is not overwritten. the origin is assumed at 0000h, an
; must be changed if the controller brings the cold start
; loader into another area, or if a read/only memory area
; is used.

```
0000          org      0          ; base of ram in cp/m
0014 =        msize    equ      20          ; min mem size in kbytes
0000 =        bias     equ      (msize-20)*1024 ; offset from 20k system
3400 =        ccp      equ      3400h+bias    ; base of the ccp
4a00 =        bios     equ      ccp+1600h      ; base of the bios
0300 =        biosl    equ      0300h         ; length of the bios
4a00 =        boot     equ      bios
1900 =        size     equ      bios+biosl-ccp ; size of cp/m system
0032 =        sects    equ      size/128      ; # of sectors to load
```

; begin the load operation

cold:

```
0000 010200    lxi      b,2          ; b=0, c=sector 2
0003 1632      mvi      d,sects      ; d=# sectors to load
0005 210034    lxi      h,ccp        ; base transfer address
```

lsect: ; load the next sector

```
; insert inline code at this point to
; read one 128 byte sector from the
; track given in register b, sector
; given in register c,
; into the address given by <hl>
;
; branch to location "cold" if a read error occurs
```

```

; *****
; *
; *      user supplied read operation goes here...
; *
; *****

0008 c36b00      jmp      past$patch      ; remove this when patche
000b             ds          60h

past$patch:
; go to next sector if load is incomplete
006b 15          dcr      d              ; sects=sects-1
006c ca004a      jz       boot           ; head for the bios

;      more sectors to load
;
; we aren't using a stack, so use <sp> as scratch registe
;      to hold the load address increment

006f 318000      lxi      sp,128          ; 128 bytes per sector
0072 39          dad      sp             ; <hl> = <hl> + 128

0073 0c          inr      c              ; sector = sector + 1
0074 79          mov      a,c
0075 felb        cpi      27            ; last sector of track?
0077 da0800      jc       lsect         ; no, go read another

; end of track, increment to next track

007a 0e01        mvi      c,1           ; sector = 1
007c 04          inr      b             ; track = track + 1
007d c30800      jmp      lsect         ; for another group
0080             end                  ; of boot loader

```


APPENDIX F: CP/M DISK DEFINITION LIBRARY

```

1: ;      CP/M 2.0 disk re-definition library
2: ;
3: ;      Copyright (c) 1979
4: ;      Digital Research
5: ;      Box 579
6: ;      Pacific Grove, CA
7: ;      93950
8: ;
9: ;      CP/M logical disk drives are defined using the
10: ;      macros given below, where the sequence of calls
11: ;      is:
12: ;
13: ;      disks      n
14: ;      diskdef parameter-list-0
15: ;      diskdef parameter-list-1
16: ;      ...
17: ;      diskdef parameter-list-n
18: ;      endef
19: ;
20: ;      where n is the number of logical disk drives attached
21: ;      to the CP/M system, and parameter-list-i defines the
22: ;      characteristics of the ith drive (i=0,1,...,n-1)
23: ;
24: ;      each parameter-list-i takes the form
25: ;          dn,fsc,lsc,[skf],bls,dks,dir,cks,ofs,[0]
26: ;      where
27: ;      dn      is the disk number 0,1,...,n-1
28: ;      fsc      is the first sector number (usually 0 or 1)
29: ;      lsc      is the last sector number on a track
30: ;      skf      is optional "skew factor" for sector translate
31: ;      bls      is the data block size (1024,2048,...,16384)
32: ;      dks      is the disk size in bls increments (word)
33: ;      dir      is the number of directory elements (word)
34: ;      cks      is the number of dir elements to checksum
35: ;      ofs      is the number of tracks to skip (word)
36: ;      [0]      is an optional 0 which forces 16K/directory en
37: ;
38: ;      for convenience, the form
39: ;          dn,dm
40: ;      defines disk dn as having the same characteristics as
41: ;      a previously defined disk dm.
42: ;
43: ;      a standard four drive CP/M system is defined by
44: ;          disks      4
45: ;          diskdef 0,1,26,6,1024,243,64,64,2
46: ;      dsk      set      0
47: ;          rept      3
48: ;      dsk      set      dsk+1
49: ;          diskdef %dsk,0
50: ;          endm
51: ;          endef
52: ;
53: ;      the value of "begdat" at the end of assembly defines t

```

```

54: ;      beginning of the uninitialize ram area above the bios,
55: ;      while the value of "enddat" defines the next location
56: ;      following the end of the data area.  the size of this
57: ;      area is given by the value of "datsiz" at the end of t
58: ;      assembly.  note that the allocation vector will be qui
59: ;      large if a large disk size is defined with a small blo
60: ;      size.
61: ;
62: dskhdr macro   dn
63: ;;           define a single disk header list
64: dpb&dn: dw      xlt&dn,0000h      ;translate table
65:         dw      0000h,0000h      ;scratch area
66:         dw      dirbuf,dpb&dn    ;dir buff,param block
67:         dw      csv&dn,alv&dn    ;check, alloc vectors
68:         endm
69: ;
70: disks macro   nd
71: ;;           define nd disks
72: ndisks set      nd                ;;for later reference
73: dpbase equ      $                ;base of disk parameter blocks
74: ;;           generate the nd elements
75: asknxt set      0
76:         rept     nd
77:         dskhdr   %asknxt
78: asknxt set      asknxt+1
79:         endm
80:         endm
81: ;
82: dpbhdr macro   dn
83: dpb&dn equ      $                ;disk parm block
84:         endm
85: ;
86: ddb macro      data,comment
87: ;;           define a db statement
88:         db      data              comment
89:         endm
90: ;
91: ddw macro      data,comment
92: ;;           define a dw statement
93:         dw      data              comment
94:         endm
95: ;
96: gcd macro      m,n
97: ;;           greatest common divisor of m,n
98: ;;           produces value gcdn as result
99: ;;           (used in sector translate table generation)
100: gcdm set      m                ;;variable for m
101: gcdn set      n                ;;variable for n
102: gcdr set      0                ;;variable for r
103:         rept     65535
104:         gcdx set      gcdm/gcdn
105:         gcdr set      gcdm - gcdx*gcdn
106:         if      gcdr = 0
107:         exitm
108:         endif

```

```

109: gcdm      set      gcdn
110: gcdn      set      gcdr
111:           endm
112:           endm
113: ;
114: diskdef macro  dn,fsc,lsc,skf,bls,dks,dir,cks,bfs,kl6
115: ;;          generate the set statements for later tables
116:           if      nul lsc
117: ;;          current disk dn same as previous fsc
118: dpb&dn equ      dpb&fsc ;equivalent parameters
119: als&dn equ      als&fsc ;same allocation vector size
120: css&dn equ      css&fsc ;same checksum vector size
121: xlt&dn equ      xlt&fsc ;same translate table
122:           else
123: secmax set      lsc-(fsc)          ;;sectors 0...secmax
124: sectors set      secmax+1;;number of sectors
125: als&dn set      (dks)/8 ;;size of allocation vector
126:           if      ((dks) mod 8) ne 0
127: als&dn set      als&dn+1
128:           endif
129: css&dn set      (cks)/4 ;;number of checksum elements
130: ;;          generate the block shift value
131: blkval set      bls/128 ;;number of sectors/block
132: blkshf set      0 ;;counts right 0's in blkval
133: blkmsk set      0 ;;fills with 1's from right
134:           rept    16 ;;once for each bit position
135:           if      blkval=1
136:           exitm
137:           endif
138: ;;          otherwise, high order 1 not found yet
139: blkshf set      blkshf+1
140: blkmsk set      (blkmsk shl 1) or 1
141: blkval set      blkval/2
142:           endm
143: ;;          generate the extent mask byte
144: blkval set      bls/1024 ;;number of kilobytes/block
145: extmsk set      0 ;;fill from right with 1's
146:           rept    16
147:           if      blkval=1
148:           exitm
149:           endif
150: ;;          otherwise more to shift
151: extmsk set      (extmsk shl 1) or 1
152: blkval set      blkval/2
153:           endm
154: ;;          may be double byte allocation
155:           if      (dks) > 256
156: extmsk set      (extmsk shr 1)
157:           endif
158: ;;          may be optional [0] in last position
159:           if      not nul kl6
160: extmsk set      kl6
161:           endif
162: ;;          now generate directory reservation bit vector
163: dirrem set      dir          ;;# remaining to process

```

```

164: dirbks    set      bls/32    ;;number of entries per block
165: dirblk    set      0         ;;fill with 1's on each loop
166:          rept     16
167:          if       dirrem=0
168:          exitm
169:          endif
170: ;;         not complete, iterate once again
171: ;;         shift right and add 1 high order bit
172: dirblk    set      (dirblk shr 1) or 8000h
173:          if       dirrem > dirbks
174: dirrem     set      dirrem-dirbks
175:          else
176: dirrem     set      0
177:          endif
178:          endm
179: dpbhdr     dn        ;;generate equ $
180:          ddw      %sectors,<;sec per track>
181:          ddb      %blkshf,<;block shift>
182:          ddb      %blkmsk,<;block mask>
183:          ddb      %extmsk,<;extnt mask>
184:          ddw      %(dks)-1,<;disk size-1>
185:          ddw      %(dir)-1,<;directory max>
186:          ddb      %dirblk shr 8,<;alloc0>
187:          ddb      %dirblk and 0ffh,<;alloc1>
188:          ddw      %(cks)/4,<;check size>
189:          ddw      %ofs,<;offset>
190: ;;         generate the translate table, if requested
191:          if       nul skf
192: xlt&dn     equ      0           ;no xlate table
193:          else
194:          if       skf = 0
195: xlt&dn     equ      0           ;no xlate table
196:          else
197: ;;         generate the translate table
198:          nxtsec   set      0     ;;next sector to fill
199:          nxtbas   set      0     ;;mcves by one on overflow
200:          gcd      %sectors,skf
201: ;;         gcdn = gcd(sectors,skew)
202:          neltst   set      sectors/gcdn
203: ;;         neltst is number of elements to generate
204: ;;         before we overlap previous elements
205:          nelts    set      neltst ;;counter
206:          xlt&dn   equ      $     ;translate table
207:          rept     sectors ;;once for each sector
208:          if       sectors < 256
209:          ddb      %nxtsec+(fsc)
210:          else
211:          ddw      %nxtsec+(fsc)
212:          endif
213:          nxtsec   set      nxtsec+(skf)
214:          if       nxtsec >= sectors
215:          nxtsec   set      nxtsec-sectors
216:          endif
217:          nelts    set      nelts-1
218:          if       nelts = 0

```

```

219: nxtbas    set      nxtbas+1
220: nxtsec    set      nxtbas
221: nelts     set      neltst
222:           endif
223:           endm
224:           endif    ;;end of nul fac test
225:           endif    ;;end of nul bls test
226:           endm
227: ;
228: defds      macro    lab,space
229: lab:        ds       space
230:           endm
231: ;
232: lds        macro    lb,dn,val
233:           defds      lb&dn,%val&dn
234:           endm
235: ;
236: endef      macro
237: ;;          generate the necessary ram data areas
238: begdat     equ       $
239: dirbuf:    ds        128      ;directory access buffer
240: dsknxt     set       0
241:           rept      ndisks    ;;once for each disk
242:           lds       alv,%dsknxt,als
243:           lds       csv,%dsknxt,css
244: dsknxt     set       dsknxt+1
245:           endm
246: enddat     equ       $
247: datsiz     equ       $-begdat
248: ;;         db 0 at this point forces hex record
249:           endm

```


APPENDIX G: BLOCKING AND DEBLOCKING ALGORITHMS.

```

1: ;*****
2: ;*
3: ;*      Sector Deblocking Algorithms for CP/M 2.0
4: ;*
5: ;*****
6: ;
7: ;      utility macro to compute sector mask
8: smask macro hblk
9: ;;      compute log2(hblk), return @x as result
10: ;;      (2 ** @x = hblk on return)
11: @y set hblk
12: @x set 0
13: ;;      count right shifts of @y until = 1
14: rept 8
15: if @y = 1
16: exitm
17: endif
18: ;;      @y is not 1, shift right one position
19: @y set @y shr 1
20: @x set @x + 1
21: endm
22: endm
23: ;
24: ;*****
25: ;*
26: ;*      CP/M to host disk constants
27: ;*
28: ;*****
29: blksiz equ 2048 ;CP/M allocation size
30: hstsiz equ 512 ;host disk sector size
31: hstspt equ 20 ;host disk sectors/trk
32: hstblk equ hstsiz/128 ;CP/M sects/host buff
33: cpmspt equ hstblk * hstspt ;CP/M sectors/track
34: secmsk equ hstblk-1 ;sector mask
35: smask equ hstblk ;compute sector mask
36: secshf equ @x ;log2(hstblk)
37: ;
38: ;*****
39: ;*
40: ;*      BDOS constants on entry to write
41: ;*
42: ;*****
43: wrall equ 0 ;write to allocated
44: wrdir equ 1 ;write to directory
45: wrual equ 2 ;write to unallocated
46: ;
47: ;*****
48: ;*
49: ;*      The BDOS entry points given below show the
50: ;*      code which is relevant to deblocking only.
51: ;*
52: ;*****
53: ;

```

```

54: ;          DISKDEF macro, or hand coded tables go here
55: dpbase    equ      $          ;disk param block base
56: ;
57: boot:
58: wboot:
59:           ;enter here on system boot to initialize
60:          xra      a          ;0 to accumulator
61:          sta      hstact      ;host buffer inactive
62:          sta      unacnt      ;clear unalloc count
63:          ret
64: ;
65: seldsk:
66:          ;select disk
67:          mov      a,c          ;selected disk number
68:          sta      sekdisk      ;seek disk number
69:          mov      l,a          ;disk number to HL
70:          mvi      h,0
71:          rept     4            ;multiply by 16
72:          dad      h
73:          endm
74:          lxi      d,dpbase      ;base of parm block
75:          dad      d            ;hl=.dpb(curdisk)
76:          ret
77: ;
78: settrk:
79:          ;set track given by registers BC
80:          mov      h,b
81:          mov      l,c
82:          shld     sektrk        ;track to seek
83:          ret
84: ;
85: setsec:
86:          ;set sector given by register c
87:          mov      a,c
88:          sta      seksec        ;sector to seek
89:          ret
90: ;
91: setdma:
92:          ;set dma address given by BC
93:          mov      h,b
94:          mov      l,c
95:          shld     dmaadr
96:          ret
97: ;
98: sectran:
99:          ;translate sector number BC
100:         mov      h,b
101:         mov      l,c
102:         ret
103: ;

```

```

104: ;*****
105: ;*
106: ;*      The READ entry point takes the place of      *
107: ;*      the previous BIOS definition for READ.      *
108: ;*
109: ;*****
110: read:
111:      ;read the selected CP/M sector
112:      mvi      a,l
113:      sta      readop      ;read operation
114:      sta      rsflag      ;must read data
115:      mvi      a,wrua1
116:      sta      wrtype      ;treat as unalloc
117:      jmp      rwoper      ;to perform the read
118: ;
119: ;*****
120: ;*
121: ;*      The WRITE entry point takes the place of      *
122: ;*      the previous BIOS definition for WRITE.      *
123: ;*
124: ;*****
125: write:
126:      ;write the selected CP/M sector
127:      xra      a      ;0 to accumulator
128:      sta      readop      ;not a read operation
129:      mov      a,c      ;write type in c
130:      sta      wrtype
131:      cpi      wrua1      ;write unallocated?
132:      jnz      chkuna      ;check for unalloc
133: ;
134: ;      write to unallocated, set parameters
135:      mvi      a,blksiz/128      ;next unalloc recs
136:      sta      unacnt
137:      lda      sekdisk      ;disk to seek
138:      sta      unadsk      ;unadsk = sekdisk
139:      lhld     sektrk
140:      shld     unatrkr      ;unatrkr = sektrk
141:      lda      seksec
142:      sta      unasec      ;unasec = seksec
143: ;
144: chkuna:
145:      ;check for write to unallocated sector
146:      lda      unacnt      ;any unalloc remain?
147:      ora      a
148:      jz      alloc      ;skip if not
149: ;
150: ;      more unallocated records remain
151:      dcr      a      ;unacnt = unacnt-1
152:      sta      unacnt
153:      lda      sekdisk      ;same disk?
154:      lxi      h,unadsk
155:      cmp      m      ;sekdisk = unadsk?
156:      jnz      alloc      ;skip if not
157: ;
158: ;      disks are the same

```

```

159:      lxi      h,unatrkl
160:      call     sektrkcmp      ;sektrk = unatrkl?
161:      jnz      alloc         ;skip if not
162: ;
163: ;      tracks are the same
164:      lda      seksec         ;same sector?
165:      lxi      h,unasecl
166:      cmp      m              ;seksec = unasecl?
167:      jnz      alloc         ;skip if not
168: ;
169: ;      match, move to next sector for future ref
170:      inr      m              ;unasecl = unasecl+1
171:      mov      a,m            ;end of track?
172:      cpi      cpmspt         ;count CP/M sectors
173:      jc       noovf          ;skip if no overflow
174: ;
175: ;      overflow to next track
176:      mvi      m,0            ;unasecl = 0
177:      lhld     unatrkl
178:      inx      h
179:      shld     unatrkl        ;unatrkl = unatrkl+1
180: ;
181: noovf:
182:      ;match found, mark as unnecessary read
183:      xra      a              ;0 to accumulator
184:      sta      rsflag         ;rsflag = 0
185:      jmp      rwoper         ;to perform the write
186: ;
187: alloc:
188:      ;not an unallocated record, requires pre-read
189:      xra      a              ;0 to accum
190:      sta      unacnt         ;unacnt = 0
191:      inr      a              ;1 to accum
192:      sta      rsflag         ;rsflag = 1
193: ;
194: ;*****
195: ;*
196: ;*      Common code for READ and WRITE follows
197: ;*
198: ;*****
199: rwoper:
200:      ;enter here to perform the read/write
201:      xra      a              ;zero to accum
202:      sta      erflag         ;no errors (yet)
203:      lda      seksec         ;compute host sector
204:      rept     secshf
205:      ora      a              ;carry = 0
206:      rar      ;shift right
207:      endm
208:      sta      sekfst         ;host sector to seek
209: ;
210: ;      active host sector?
211:      lxi      h,hstact       ;host active flag
212:      mov      a,m
213:      mvi      m,1            ;always becomes 1

```

```

214:      ora      a      ;was it already?
215:      jz       filhst  ;fill host if not
216: ;
217: ;      host buffer active, same as seek buffer?
218:      lda      sekdisk
219:      lxi      h,hstdsk      ;same disk?
220:      cmp      m      ;sekdisk = hstdsk?
221:      jnz      nomatch
222: ;
223: ;      same disk, same track?
224:      lxi      h,hsttrk
225:      call     sektrkcmp      ;sektrk = hsttrk?
226:      jnz      nomatch
227: ;
228: ;      same disk, same track, same buffer?
229:      lda      sekhst
230:      lxi      h,hstsec      ;sekhst = hstsec?
231:      cmp      m
232:      jz       match      ;skip if match
233: ;
234: nomatch:
235:      ;proper disk, but not correct sector
236:      lda      hstwrtr      ;host written?
237:      ora      a
238:      cnz      writehst      ;clear host buff
239: ;
240: filhst:
241:      ;may have to fill the host buffer
242:      lda      sekdisk
243:      sta      hstdsk
244:      lhld     sektrk
245:      shld     hsttrk
246:      lda      sekhst
247:      sta      hstsec
248:      lda      rsflag      ;need to read?
249:      ora      a
250:      cnz      readhst      ;yes, if 1
251:      xra      a      ;0 to accum
252:      sta      hstwrtr      ;no pending write
253: ;
254: match:
255:      ;copy data to or from buffer
256:      lda      seksec      ;mask buffer number
257:      ani      secmsk      ;least signif bits
258:      mov      l,a      ;ready to shift
259:      mvi      h,0      ;double count
260:      rept     7      ;shift left 7
261:      dad      h
262:      endm
263: ;      hl has relative host buffer address
264:      lxi      d,hstbuf
265:      dad      d      ;hl = host address
266:      xchg      ;now in DE
267:      lhld     dmaadr      ;get/put CP/M data
268:      mvi      c,128      ;length of move

```



```

269:      lda      readop      ;which way?
270:      ora      a
271:      jnz      rwmov      ;skip if read
272: ;
273: ;      write operation, mark and switch direction
274:      mvi      a,1
275:      sta      hstwr      ;hstwr = 1
276:      xchg
277: ;
278: rwmov:
279:      ;C initially 128, DE is source, HL is dest
280:      ldax     d      ;source character
281:      inx      d
282:      mov      m,a      ;to dest
283:      inx      h
284:      dcr      c      ;loop 128 times
285:      jnz      rwmov
286: ;
287: ;      data has been moved to/from host buffer
288:      lda      wrtype      ;write type
289:      cpi      wrdir      ;to directory?
290:      lda      erflag      ;in case of errors
291:      rnz
292: ;
293: ;      clear host buffer for directory write
294:      ora      a      ;errors?
295:      rnz      ;skip if so
296:      xra      a      ;0 to accum
297:      sta      hstwr      ;buffer written
298:      call     writehst
299:      lda      erflag
300:      ret
301: ;
302: ;*****
303: ;*
304: ;*      Utility subroutine for 16-bit compare
305: ;*
306: ;*****
307: sektrkcmp:
308:      ;HL = .unatr      or .hsttr, compare with sektr
309:      xchg
310:      lxi      h,sektr
311:      ldax     d      ;low byte compare
312:      cmp      m      ;same?
313:      rnz      ;return if not
314: ;      low bytes equal, test high ls
315:      inx      d
316:      inx      h
317:      ldax     d
318:      cmp      m      ;sets flags
319:      ret
320: ;

```

```

321: ;*****
322: ;*
323: ;*      WRITEHST performs the physical write to      *
324: ;*      the host disk, READHST reads the physical    *
325: ;*      disk.                                         *
326: ;*
327: ;*****
328: writehst:
329:      ;hstdsk = host disk #, hsttrk = host track #,
330:      ;hstsec = host sect #. write "hstsiz" bytes
331:      ;from hstbuf and return error flag in erflag.
332:      ;return erflag non-zero if error
333:      ret
334: ;
335: readhst:
336:      ;hstdsk = host disk #, hsttrk = host track #,
337:      ;hstsec = host sect #. read "hstsiz" bytes
338:      ;into hstbuf and return error flag in erflag.
339:      ret
340: ;
341: ;*****
342: ;*
343: ;*      Unitialized RAM data areas
344: ;*
345: ;*****
346: ;
347: sekdisk: ds      1          ;seek disk number
348: sektrk: ds      2          ;seek track number
349: seksec: ds      1          ;seek sector number
350: ;
351: hstdisk: ds      1          ;host disk number
352: hsttrk: ds      2          ;host track number
353: hstsec: ds      1          ;host sector number
354: ;
355: sekhst: ds      1          ;seek shr secshf
356: hstact: ds      1          ;host active flag
357: hstwrt: ds      1          ;host written flag
358: ;
359: unacnt: ds      1          ;unalloc rec cnt
360: unadsk: ds      1          ;last unalloc disk
361: unatrk: ds      2          ;last unalloc track
362: unasec: ds      1          ;last unalloc sector
363: ;
364: erflag: ds      1          ;error reporting
365: rsflag: ds      1          ;read sector flag
366: readop: ds      1          ;1 if read operation
367: wrtype: ds      1          ;write operation type
368: dmaadr: ds      2          ;last dma address
369: hstbuf: ds      hstsiz     ;host buffer
370: ;

```

```
371: ;*****
372: ;*
373: ;*      The ENDEF macro invocation goes here
374: ;*
375: ;*****
376:      end
```




Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M 2.2 INTERFACE GUIDE

Copyright (c) 1979

DIGITAL RESEARCH

Copyright (c) 1979 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

CP/M 2.2 INTERFACE GUIDE

Copyright (c) 1979
Digital Research, Box 579
Pacific Grove, California

1. Introduction	1
2. Operating System Call Conventions	3
3. A Sample File-to-File Copy Program	29
4. A Sample File Dump Utility	34
5. A Sample Random Access Program	37
6. System Function Summary	46

THE 7-17-1968

THE 7-17-1968

THE 7-17-1968

THE 7-17-1968

THE 7-17-1968

THE 7-17-1968

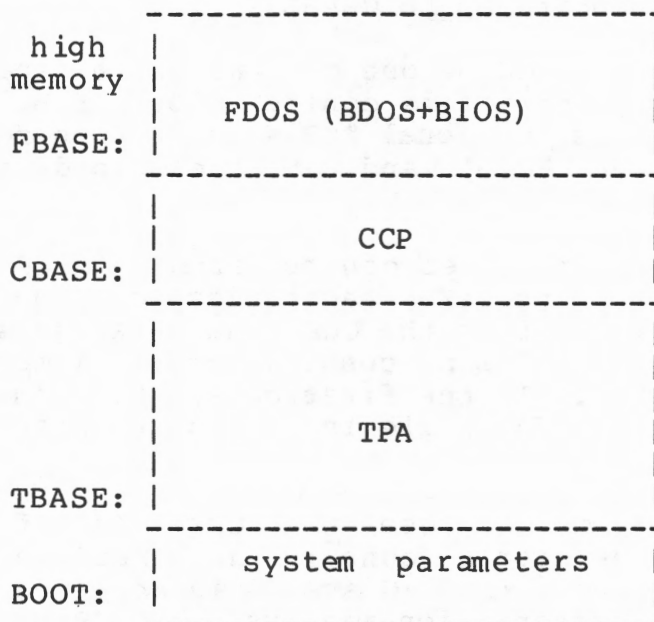
THE 7-17-1968

THE 7-17-1968

1. INTRODUCTION.

This manual describes CP/M, release 2, system organization including the structure of memory and system entry points. The intention is to provide the necessary information required to write programs which operate under CP/M, and which use the peripheral and disk I/O facilities of the system.

CP/M is logically divided into four parts, called the Basic I/O System (BIOS), the Basic Disk Operating System (BDOS), the Console command processor (CCP), and the Transient Program Area (TPA). The BIOS is a hardware-dependent module which defines the exact low level interface to a particular computer system which is necessary for peripheral device I/O. Although a standard BIOS is supplied by Digital Research, explicit instructions are provided for field reconfiguration of the BIOS to match nearly any hardware environment (see the Digital Research manual entitled "CP/M Alteration Guide"). The BIOS and BDOS are logically combined into a single module with a common entry point, and referred to as the FDOS. The CCP is a distinct program which uses the FDOS to provide a human-oriented interface to the information which is cataloged on the backup storage device. The TPA is an area of memory (i.e., the portion which is not used by the FDOS and CCP) where various non-resident operating system commands and user programs are executed. The lower portion of memory is reserved for system information and is detailed later sections. Memory organization of the CP/M system is shown below:



The exact memory addresses corresponding to BOOT, TBASE, CBASE, and FBASE vary from version to version, and are described fully in the "CP/M Alteration Guide." All standard CP/M versions, however, assume BOOT = 0000H, which is the base of random access memory. The machine code found at location BOOT performs a system "warm start" which loads and initializes the programs and variables necessary to return control to the CCP. Thus, transient programs need only jump to location BOOT

(All Information Contained Herein is Proprietary to Digital Research.)

to return control to CP/M at the command level. Further, the standard versions assume TBASE = BOOT+0100H which is normally location 0100H. The principal entry point to the FDOS is at location BOOT+0005H (normally 0005H) where a jump to FBASE is found. The address field at BOOT+0006H (normally 0006H) contains the value of FBASE and can be used to determine the size of available memory, assuming the CCP is being overlayed by a transient program.

Transient programs are loaded into the TPA and executed as follows. The operator communicates with the CCP by typing command lines following each prompt. Each command line takes one of the forms:

```
command
command file1
command file1 file2
```

where "command" is either a built-in function such as DIR or TYPE, or the name of a transient command or program. If the command is a built-in function of CP/M, it is executed immediately. Otherwise, the CCP searches the currently addressed disk for a file by the name

command.COM

If the file is found, it is assumed to be a memory image of a program which executes in the TPA, and thus implicitly originates at TBASE in memory. The CCP loads the COM file from the disk into memory starting at TBASE and possibly extending up to CBASE.

If the command is followed by one or two file specifications, the CCP prepares one or two file control block (FCB) names in the system parameter area. These optional FCB's are in the form necessary to access files through the FDOS, and are described in the next section.

The transient program receives control from the CCP and begins execution, perhaps using the I/O facilities of the FDOS. The transient program is "called" from the CCP, and thus can simply return to the CCP upon completion of its processing, or can jump to BOOT to pass control back to CP/M. In the first case, the transient program must not use memory above CBASE, while in the latter case, memory up through FBASE-1 is free.

The transient program may use the CP/M I/O facilities to communicate with the operator's console and peripheral devices, including the disk subsystem. The I/O system is accessed by passing a "function number" and an "information address" to CP/M through the FDOS entry point at BOOT+0005H. In the case of a disk read, for example, the transient program sends the number corresponding to a disk read, along with the address of an FCB to the CP/M FDOS. The FDOS, in turn, performs the operation and returns with either a disk read completion indication or an error number indicating that the disk read was unsuccessful. The function numbers and error indicators are given in below.

(All Information Contained Herein is Proprietary to Digital Research.)

2. OPERATING SYSTEM CALL CONVENTIONS.

The purpose of this section is to provide detailed information for performing direct operating system calls from user programs. Many of the functions listed below, however, are more simply accessed through the I/O macro library provided with the MAC macro assembler, and listed in the Digital Research manual entitled "MAC Macro Assembler: Language Manual and Applications Guide."

CP/M facilities which are available for access by transient programs fall into two general categories: simple device I/O, and disk file I/O. The simple device operations include:

- Read a Console Character
- Write a Console Character
- Read a Sequential Tape Character
- Write a Sequential Tape Character
- Write a List Device Character
- Get or Set I/O Status
- Print Console Buffer
- Read Console Buffer
- Interrogate Console Ready

The FDOS operations which perform disk Input/Output are

- Disk System Reset
- Drive Selection
- File Creation
- File Open
- File Close
- Directory Search
- File Delete
- File Rename
- Random or Sequential Read
- Random or Sequential Write
- Interrogate Available Disks
- Interrogate Selected Disk
- Set DMA Address
- Set/Reset File Indicators

As mentioned above, access to the FDOS functions is accomplished by passing a function number and information address through the primary entry point at location `BOOT+0005H`. In general, the function number is passed in register C with the information address in the double byte pair DE. Single byte values are returned in register A, with double byte values returned in HL (a zero value is returned when the function number is out of range). For reasons of compatibility, register A = L and register B = H upon return in all cases. Note that the register passing conventions of CP/M agree with those of Intel's PL/M systems programming language. The list of CP/M function numbers is given below.

(All Information Contained Herein is Proprietary to Digital Research.)

0	System Reset	19	Delete File
1	Console Input	20	Read Sequential
2	Console Output	21	Write Sequential
3	Reader Input	22	Make File
4	Punch Output	23	Rename File
5	List Output	24	Return Login Vector
6	Direct Console I/O	25	Return Current Disk
7	Get I/O Byte	26	Set DMA Address
8	Set I/O Byte	27	Get Addr(Alloc)
9	Print String	28	Write Protect Disk
10	Read Console Buffer	29	Get R/O Vector
11	Get Console Status	30	Set File Attributes
12	Return Version Number	31	Get Addr(Disk Parms)
13	Reset Disk System	32	Set/Get User Code
14	Select Disk	33	Read Random
15	Open File	34	Write Random
16	Close File	35	Compute File Size
17	Search for First	36	Set Random Record
18	Search for Next		

(Functions 28 and 32 should be avoided in application programs to maintain upward compatibility with MP/M.)

Upon entry to a transient program, the CCP leaves the stack pointer set to an eight level stack area with the CCP return address pushed onto the stack, leaving seven levels before overflow occurs. Although this stack is usually not used by a transient program (i.e., most transients return to the CCP though a jump to location 0000H), it is sufficiently large to make CP/M system calls since the FDOS switches to a local stack at system entry. The following assembly language program segment, for example, reads characters continuously until an asterisk is encountered, at which time control returns to the CCP (assuming a standard CP/M system with BOOT = 0000H):

```

BDOS    EQU    0005H    ;STANDARD CP/M ENTRY
CONIN   EQU    1        ;CONSOLE INPUT FUNCTION
;
NEXTC:  ORG     0100H    ;BASE OF TPA
        MVI     C,CONIN ;READ NEXT CHARACTER
        CALL    BDOS    ;RETURN CHARACTER IN <A>
        CPI     '*'     ;END OF PROCESSING?
        JNZ     NEXTC   ;LOOP IF NOT
        RET      ;RETURN TO CCP
        END

```

CP/M implements a named file structure on each disk, providing a logical organization which allows any particular file to contain any number of records from completely empty, to the full capacity of the drive. Each drive is logically distinct with a disk directory and file data area. The disk file names are in three parts: the drive select code, the file name consisting of one to eight non-blank characters, and the file type consisting of zero to three non-blank characters. The file type names the generic category of a particular file, while the file name distinguishes individual files in each category. The file types listed below name a few generic categories

(All Information Contained Herein is Proprietary to Digital Research.)

which have been established, although they are generally arbitrary:

ASM	Assembler Source	PLI	PL/I Source File
PRN	Printer Listing	REL	Relocatable Module
HEX	Hex Machine Code	TEX	TEX Formatter Source
BAS	Basic Source File	BAK	ED Source Backup
INT	Intermediate Code	SYM	SID Symbol File
COM	CCP Command File	\$\$\$	Temporary File

Source files are treated as a sequence of ASCII characters, where each "line" of the source file is followed by a carriage-return line-feed sequence (0DH followed by 0AH). Thus one 128 byte CP/M record could contain several lines of source text. The end of an ASCII file is denoted by a control-Z character (1AH) or a real end of file, returned by the CP/M read operation. Control-Z characters embedded within machine code files (e.g., COM files) are ignored, however, and the end of file condition returned by CP/M is used to terminate read operations.

Files in CP/M can be thought of as a sequence of up to 65536 records of 128 bytes each, numbered from 0 through 65535, thus allowing a maximum of 8 megabytes per file. Note, however, that although the records may be considered logically contiguous, they may not be physically contiguous in the disk data area. Internally, all files are broken into 16K byte segments called logical extents, so that counters are easily maintained as 8-bit values. Although the decomposition into extents is discussed in the paragraphs which follow, they are of no particular consequence to the programmer since each extent is automatically accessed in both sequential and random access modes.

In the file operations starting with function number 15, DE usually addresses a file control block (FCB). Transient programs often use the default file control block area reserved by CP/M at location BOOT+005CH (normally 005CH) for simple file operations. The basic unit of file information is a 128 byte record used for all file operations, thus a default location for disk I/O is provided by CP/M at location BOOT+0080H (normally 0080H) which is the initial default DMA address (see function 26). All directory operations take place in a reserved area which does not affect write buffers as was the case in release 1, with the exception of Search First and Search Next, where compatibility is required.

The File Control Block (FCB) data area consists of a sequence of 33 bytes for sequential access and a series of 36 bytes in the case that the file is accessed randomly. The default file control block normally located at 005CH can be used for random access files, since the three bytes starting at BOOT+007DH are available for this purpose. The FCB format is shown with the following fields:

```

-----
|dr|f1|f2|/ /|f8|t1|t2|t3|ex|s1|s2|rc|d0|/ /|dn|cr|r0|rl|r2|
-----
00 01 02 ... 08 09 10 11 12 13 14 15 16 ... 31 32 33 34 35

```

where

dr drive code (0 - 16)
 0 => use default drive for file
 1 => auto disk select drive A,
 2 => auto disk select drive B,
 ...
 16=> auto disk select drive P.

f1...f8 contain the file name in ASCII
 upper case, with high bit = 0

t1,t2,t3 contain the file type in ASCII
 upper case, with high bit = 0
 t1', t2', and t3' denote the
 bit of these positions,
 t1' = 1 => Read/Only file,
 t2' = 1 => SYS file, no DIR list

ex contains the current extent number,
 normally set to 00 by the user, but
 in range 0 - 31 during file I/O

s1 reserved for internal system use

s2 reserved for internal system use, set
 to zero on call to OPEN, MAKE, SEARCH

rc record count for extent "ex,"
 takes on values from 0 - 128

d0...dn filled-in by CP/M, reserved for
 system use

cr current record to read or write in
 a sequential file operation, normally
 set to zero by user

r0,rl,r2 optional random record number in the
 range 0-65535, with overflow to r2,
 r0,rl constitute a 16-bit value with
 low byte r0, and high byte rl

Each file being accessed through CP/M must have a corresponding FCB which provides the name and allocation information for all subsequent file operations. When accessing files, it is the programmer's responsibility to fill the lower sixteen bytes of the FCB and initialize the "cr" field. Normally, bytes 1 through 11 are set to the ASCII character values for the file name and file type, while all other fields are zero.

(All Information Contained Herein is Proprietary to Digital Research.)

FCB's are stored in a directory area of the disk, and are brought into central memory before proceeding with file operations (see the OPEN and MAKE functions). The memory copy of the FCB is updated as file operations take place and later recorded permanently on disk at the termination of the file operation (see the CLOSE command).

The CCP constructs the first sixteen bytes of two optional FCB's for a transient by scanning the remainder of the line following the transient name, denoted by "file1" and "file2" in the prototype command line described above, with unspecified fields set to ASCII blanks. The first FCB is constructed at location BOOT+005CH, and can be used as-is for subsequent file operations. The second FCB occupies the d0 ... dn portion of the first FCB, and must be moved to another area of memory before use. If, for example, the operator types

PROGNAME B:X.ZOT Y.ZAP

the file PROGNAME.COM is loaded into the TPA, and the default FCB at BOOT+005CH is initialized to drive code 2, file name "X" and file type "ZOT". The second drive code takes the default value 0, which is placed at BOOT+006CH, with the file name "Y" placed into location BOOT+006DH and file type "ZAP" located 8 bytes later at BOOT+0075H. All remaining fields through "cr" are set to zero. Note again that it is the programmer's responsibility to move this second file name and type to another area, usually a separate file control block, before opening the file which begins at BOOT+005CH, due to the fact that the open operation will overwrite the second name and type.

If no file names are specified in the original command, then the fields beginning at BOOT+005DH and BOOT+006DH contain blanks. In all cases, the CCP translates lower case alphabetics to upper case to be consistent with the CP/M file naming conventions.

As an added convenience, the default buffer area at location BOOT+0080H is initialized to the command line tail typed by the operator following the program name. The first position contains the number of characters, with the characters themselves following the character count. Given the above command line, the area beginning at BOOT+0080H is initialized as follows:

```
BOOT+0080H:
+00 +01 +02 +03 +04 +05 +06 +07 +08 +09 +10 +11 +12 +13 +14
14 " " "B" ":" "X" "." "Z" "O" "T" " " "Y" "." "Z" "A" "P"
```

where the characters are translated to upper case ASCII with uninitialized memory following the last valid character. Again, it is the responsibility of the programmer to extract the information from this buffer before any file operations are performed, unless the default DMA address is explicitly changed.

The individual functions are described in detail in the pages which follow.

(All Information Contained Herein is Proprietary to Digital Research.)


```

*****
*
*  FUNCTION 0:  System Reset
*
*****
*  Entry Parameters:
*    Register  C:  00H
*
*****

```

The system reset function returns control to the CP/M operating system at the CCP level. The CCP re-initializes the disk subsystem by selecting and logging-in disk drive A. This function has exactly the same effect as a jump to location BOOT.

```

*****
*
*  FUNCTION 1:  CONSOLE INPUT
*
*****
*  Entry Parameters:
*    Register  C:  01H
*
*  Returned Value:
*    Register  A:  ASCII Character
*
*****

```

The console input function reads the next console character to register A. Graphic characters, along with carriage return, line feed, and backspace (ctl-H) are echoed to the console. Tab characters (ctl-I) are expanded in columns of eight characters. A check is made for start/stop scroll (ctl-S) and start/stop printer echo (ctl-P). The FDOS does not return to the calling program until a character has been typed, thus suspending execution if a character is not ready.

```

*****
*
*  FUNCTION 2:  CONSOLE OUTPUT
*
*****
*  Entry Parameters:
*    Register  C:  02H
*    Register  E:  ASCII Character
*
*****

```

The ASCII character from register E is sent to the console device. Similar to function 1, tabs are expanded and checks are made for start/stop scroll and printer echo.

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 3:  READER INPUT
*
*****
* Entry Parameters:
*   Register   C:  03H
*
* Returned   Value:
*   Register   A:  ASCII Character
*****

```

The Reader Input function reads the next character from the logical reader into register A (see the IOBYTE definition in the "CP/M Alteration Guide"). Control does not return until the character has been read.

```

*****
*
* FUNCTION 4:  PUNCH OUTPUT
*
*****
* Entry Parameters:
*   Register   C:  04H
*   Register   E:  ASCII Character
*****

```

The Punch Output function sends the character from register E to the logical punch device.

```

*****
*
* FUNCTION 5:  LIST OUTPUT
*
*****
* Entry Parameters:
*   Register   C:  05H
*   Register   E:  ASCII Character
*****

```

The List Output function sends the ASCII character in register E to the logical listing device.

```

*****
*                                     *
*  FUNCTION 6:  DIRECT CONSOLE I/O  *
*                                     *
*****
*  Entry Parameters:                *
*      Register   C:  06H           *
*      Register   E:  0FFH (input) or *
*                      char (output)  *
*                                     *
*  Returned Value:                  *
*      Register   A:  char or status *
*                      (no value)    *
*****

```

Direct console I/O is supported under CP/M for those specialized applications where unadorned console input and output is required. Use of this function should, in general, be avoided since it bypasses all of CP/M's normal control character functions (e.g., control-S and control-P). Programs which perform direct I/O through the BIOS under previous releases of CP/M, however, should be changed to use direct I/O under BDOS so that they can be fully supported under future releases of MP/M and CP/M.

Upon entry to function 6, register E either contains hexadecimal FF, denoting a console input request, or register E contains an ASCII character. If the input value is FF, then function 6 returns A = 00 if no character is ready, otherwise A contains the next console input character.

If the input value in E is not FF, then function 6 assumes that E contains a valid ASCII character which is sent to the console.

```

*****
*                                     *
*  FUNCTION 7:  GET I/O BYTE         *
*                                     *
*****
*  Entry Parameters:                 *
*      Register   C:  07H           *
*                                     *
*  Returned   Value:                 *
*      Register   A:  I/O Byte Value *
*****

```

The Get I/O Byte function returns the current value of IOBYTE in register A. See the "CP/M Alteration Guide" for IOBYTE definition.

```

*****
*                                     *
*  FUNCTION 8:  SET I/O BYTE         *
*                                     *
*****
*  Entry Parameters:                 *
*      Register   C:  08H           *
*      Register   E:  I/O Byte Value *
*                                     *
*****

```

The Set I/O Byte function changes the system IOBYTE value to that given in register E.

```

*****
*                                     *
*  FUNCTION 9:  PRINT STRING         *
*                                     *
*****
*  Entry Parameters:                 *
*      Register   C:  09H           *
*      Registers DE: String Address *
*                                     *
*****

```

The Print String function sends the character string stored in memory at the location given by DE to the console device, until a "\$" is encountered in the string. Tabs are expanded as in function 2, and checks are made for start/stop scroll and printer echo.

```

*****
*
* FUNCTION 10: READ CONSOLE BUFFER
*
*****
* Entry Parameters:
*   Register C: 0AH
*   Registers DE: Buffer Address
*
* Returned Value:
*   Console Characters in Buffer
*****

```

The Read Buffer function reads a line of edited console input into a buffer addressed by registers DE. Console input is terminated when either the input buffer overflows. The Read Buffer takes the form:

```

DE: +0 +1 +2 +3 +4 +5 +6 +7 +8   . . .   +n
-----
|mx|nc|c1|c2|c3|c4|c5|c6|c7|   . . .   |??|
-----

```

where "mx" is the maximum number of characters which the buffer will hold (1 to 255), "nc" is the number of characters read (set by FDOS upon return), followed by the characters read from the console. if nc < mx, then uninitialized positions follow the last character, denoted by "??" in the above figure. A number of control functions are recognized during line editing:

```

rub/del removes and echoes the last character
ctl-C  reboots when at the beginning of line
ctl-E  causes physical end of line
ctl-H  backspaces one character position
ctl-J  (line feed) terminates input line
ctl-M  (return) terminates input line
ctl-R  retypes the current line after new line
ctl-U  removes currnt line after new line
ctl-X  backspaces to beginning of current line

```

Note also that certain functions which return the carriage to the leftmost position (e.g., ctl-X) do so only to the column position where the prompt ended (in earlier releases, the carriage returned to the extreme left margin). This convention makes operator data input and line correction more legible.


```

*****
*
* FUNCTION 11: GET CONSOLE STATUS
*
*****
* Entry Parameters:
*   Register C: 0BH
*
* Returned Value:
*   Register A: Console Status
*****

```

The Console Status function checks to see if a character has been typed at the console. If a character is ready, the value 0FFH is returned in register A. Otherwise a 00H value is returned.

```

*****
*
* FUNCTION 12: RETURN VERSION NUMBER
*
*****
* Entry Parameters:
*   Register C: 0CH
*
* Returned Value:
*   Registers HL: Version Number
*****

```

Function 12 provides information which allows version independent programming. A two-byte value is returned, with H = 00 designating the CP/M release (H = 01 for MP/M), and L = 00 for all releases previous to 2.0. CP/M 2.0 returns a hexadecimal 20 in register L, with subsequent version 2 releases in the hexadecimal range 21, 22, through 2F. Using function 12, for example, you can write application programs which provide both sequential and random access functions, with random access disabled when operating under early releases of CP/M.

```

*****
*
* FUNCTION 13: RESET DISK SYSTEM
*
*****
* Entry Parameters:
*   Register   C:  0DH
*
*****

```

The Reset Disk Function is used to programmatically restore the file system to a reset state where all disks are set to read/write (see functions 28 and 29), only disk drive A is selected, and the default DMA address is reset to BOOT+0080H. This function can be used, for example, by an application program which requires a disk change without a system reboot.

```

*****
*
* FUNCTION 14: SELECT DISK
*
*****
* Entry Parameters:
*   Register   C:  0EH
*   Register   E:  Selected Disk
*
*****

```

The Select Disk function designates the disk drive named in register E as the default disk for subsequent file operations, with E = 0 for drive A, 1 for drive B, and so-forth through 15 corresponding to drive P in a full sixteen drive system. The drive is placed in an "on-line" status which, in particular, activates its directory until the next cold start, warm start, or disk system reset operation. If the disk media is changed while it is on-line, the drive automatically goes to a read/only status in a standard CP/M environment (see function 28). FCB's which specify drive code zero (dr = 00H) automatically reference the currently selected default drive. Drive code values between 1 and 16, however, ignore the selected default drive and directly reference drives A through P.

```

*****
*
* FUNCTION 15: OPEN FILE
*
*****
* Entry Parameters:
*   Register C: 0FH
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

The Open File operation is used to activate a file which currently exists in the disk directory for the currently active user number. The FDOS scans the referenced disk directory for a match in positions 1 through 14 of the FCB referenced by DE (byte s1 is automatically zeroed), where an ASCII question mark (3FH) matches any directory character in any of these positions. Normally, no question marks are included and, further, bytes "ex" and "s2" of the FCB are zero.

If a directory element is matched, the relevant directory information is copied into bytes d0 through dn of the FCB, thus allowing access to the files through subsequent read and write operations. Note that an existing file must not be accessed until a successful open operation is completed. Upon return, the open function returns a "directory code" with the value 0 through 3 if the open was successful, or 0FFH (255 decimal) if the file cannot be found. If question marks occur in the FCB then the first matching FCB is activated. Note that the current record ("cr") must be zeroed by the program if the file is to be accessed sequentially from the first record.

```

*****
*
* FUNCTION 16: CLOSE FILE
*
*****
* Entry Parameters:
*   Register C: 10H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

The Close File function performs the inverse of the open file function. Given that the FCB addressed by DE has been previously activated through an open or make function (see functions 15 and 22), the close function permanently records the new FCB in the referenced disk directory. The FCB matching process for the close is identical to the open function. The directory code returned for a successful close operation is 0, 1, 2, or 3, while a 0FFH (255 decimal) is returned if the file name cannot be found in the directory. A file need not be closed if only read operations have taken place. If write operations have occurred, however, the close operation is necessary to permanently record the new directory information.

```

*****
*
* FUNCTION 17: SEARCH FOR FIRST
*
*****
* Entry Parameters:
*   Register C: 11H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

Search First scans the directory for a match with the file given by the FCB addressed by DE. The value 255 (hexadecimal FF) is returned if the file is not found, otherwise 0, 1, 2, or 3 is returned indicating the file is present. In the case that the file is found, the current DMA address is filled with the record containing the directory entry, and the relative starting position is A * 32 (i.e., rotate the A register left 5 bits, or ADD A five times). Although not normally required for application programs, the directory information can be extracted from the buffer at this position.

An ASCII question mark (63 decimal, 3F hexadecimal) in any position from "fl" through "ex" matches the corresponding field of any directory entry on the default or auto-selected disk drive. If the "dr" field contains an ASCII question mark, then the auto disk select function is disabled, the default disk is searched, with the search function returning any matched entry, allocated or free, belonging to any user number. This latter function is not normally used by application programs, but does allow complete flexibility to scan all current directory values. If the "dr" field is not a question mark, the "s2" byte is automatically zeroed.

```

*****
*
* FUNCTION 18: SEARCH FOR NEXT
*
*****
* Entry Parameters:
*   Register C: 12H
*
* Returned Value:
*   Register A: Directory Code
*****

```

The Search Next function is similar to the Search First function, except that the directory scan continues from the last matched entry. Similar to function 17, function 18 returns the decimal value 255 in A when no more directory items match.

(All Information Contained Herein is Proprietary to Digital Research.)


```

*****
*                                     *
*  FUNCTION 19: DELETE FILE          *
*                                     *
*****
*  Entry Parameters:                *
*      Register   C:  13H           *
*      Registers DE: FCB Address    *
*                                     *
*  Returned Value:                  *
*      Register   A:  Directory Code *
*****

```

The Delete File function removes files which match the FCB addressed by DE. The filename and type may contain ambiguous references (i.e., question marks in various positions), but the drive select code cannot be ambiguous, as in the Search and Search Next functions.

Function 19 returns a decimal 255 if the referenced file or files cannot be found, otherwise a value in the range 0 to 3 is returned.

```

*****
*                                     *
*  FUNCTION 20: READ SEQUENTIAL      *
*                                     *
*****
*  Entry Parameters:                *
*      Register   C:  14H           *
*      Registers DE: FCB Address    *
*                                     *
*  Returned Value:                  *
*      Register   A:  Directory Code *
*****

```

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Read Sequential function reads the next 128 byte record from the file into memory at the current DMA address. the record is read from position "cr" of the extent, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next read operation. The value 00H is returned in the A register if the read operation was successful, while a non-zero value is returned if no data exists at the next record position (e.g., end of file occurs).

```

*****
*
* FUNCTION 21: WRITE SEQUENTIAL
*
*****
* Entry Parameters:
*   Register C: 15H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

Given that the FCB addressed by DE has been activated through an open or make function (numbers 15 and 22), the Write Sequential function writes the 128 byte data record at the current DMA address to the file named by the FCB. the record is placed at position "cr" of the file, and the "cr" field is automatically incremented to the next record position. If the "cr" field overflows then the next logical extent is automatically opened and the "cr" field is reset to zero in preparation for the next write operation. Write operations can take place into an existing file, in which case newly written records overlay those which already exist in the file. Register A = 00H upon return from a successful write operation, while a non-zero value indicates an unsuccessful write due to a full disk.

```

*****
*
* FUNCTION 22: MAKE FILE
*
*****
* Entry Parameters:
*   Register C: 16H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

The Make File operation is similar to the open file operation except that the FCB must name a file which does not exist in the currently referenced disk directory (i.e., the one named explicitly by a non-zero "dr" code, or the default disk if "dr" is zero). The FDOS creates the file and initializes both the directory and main memory value to an empty file. The programmer must ensure that no duplicate file names occur, and a preceding delete operation is sufficient if there is any possibility of duplication. Upon return, register A = 0, 1, 2, or 3 if the operation was successful and 0FFH (255 decimal) if no more directory space is available. The make function has the side-effect of activating the FCB and thus a subsequent open is not necessary.

(All Information Contained Herein is Proprietary to Digital Research.)

```

*****
*
* FUNCTION 23: RENAME FILE
*
*****
* Entry Parameters:
*   Register C: 17H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Directory Code
*****

```

The Rename function uses the FCB addressed by DE to change all occurrences of the file named in the first 16 bytes to the file named in the second 16 bytes. The drive code "dr" at position 0 is used to select the drive, while the drive code for the new file name at position 16 of the FCB is assumed to be zero. Upon return, register A is set to a value between 0 and 3 if the rename was successful, and 0FFH (255 decimal) if the first file name could not be found in the directory scan.

```

*****
*
* FUNCTION 24: RETURN LOGIN VECTOR
*
*****
* Entry Parameters:
*   Register C: 18H
*
* Returned Value:
*   Registers HL: Login Vector
*****

```

The login vector value returned by CP/M is a 16-bit value in HL, where the least significant bit of L corresponds to the first drive A, and the high order bit of H corresponds to the sixteenth drive, labelled P. A "0" bit indicates that the drive is not on-line, while a "1" bit marks an drive that is actively on-line due to an explicit disk drive selection, or an implicit drive select caused by a file operation which specified a non-zero "dr" field. Note that compatibility is maintained with earlier releases, since registers A and L contain the same values upon return.

```

*****
*
* FUNCTION 25: RETURN CURRENT DISK *
*
*****
* Entry Parameters: *
*   Register C: 19H *
*
* Returned Value: *
*   Register A: Current Disk *
*****

```

Function 25 returns the currently selected default disk number in register A. The disk numbers range from 0 through 15 corresponding to drives A through P.

```

*****
*
* FUNCTION 26: SET DMA ADDRESS *
*
*****
* Entry Parameters: *
*   Register C: 1AH *
*   Registers DE: DMA Address *
*
*****

```

"DMA" is an acronym for Direct Memory Address, which is often used in connection with disk controllers which directly access the memory of the mainframe computer to transfer data to and from the disk subsystem. Although many computer systems use non-DMA access (i.e., the data is transferred through programmed I/O operations), the DMA address has, in CP/M, come to mean the address at which the 128 byte data record resides before a disk write and after a disk read. Upon cold start, warm start, or disk system reset, the DMA address is automatically set to BOOT+0080H. The Set DMA function, however, can be used to change this default value to address another area of memory where the data records reside. Thus, the DMA address becomes the value specified by DE until it is changed by a subsequent Set DMA function, cold start, warm start, or disk system reset.

```

*****
*
*  FUNCTION 27: GET ADDR(ALLOC)
*
*****
*  Entry Parameters:
*    Register C: 1BH
*
*  Returned Value:
*    Registers HL: ALLOC Address
*****

```

An "allocation vector" is maintained in main memory for each on-line disk drive. Various system programs use the information provided by the allocation vector to determine the amount of remaining storage (see the STAT program). Function 27 returns the base address of the allocation vector for the currently selected disk drive. The allocation information may, however, be invalid if the selected disk has been marked read/only. Although this function is not normally used by application programs, additional details of the allocation vector are found in the "CP/M Alteration Guide."

```

*****
*
*  FUNCTION 28: WRITE PROTECT DISK
*
*****
*  Entry Parameters:
*    Register C: 1CH
*
*****

```

The disk write protect function provides temporary write protection for the currently selected disk. Any attempt to write to the disk, before the next cold or warm start operation produces the message

Bdos Err on d: R/O


```

*****
*
* FUNCTION 29: GET READ/ONLY VECTOR *
*
*****
* Entry Parameters: *
* Register C: 1DH *
*
* Returned Value: *
* Registers HL: R/O Vector Value*
*****

```

Function 29 returns a bit vector in register pair HL which indicates drives which have the temporary read/only bit set. Similar to function 24, the least significant bit corresponds to drive A, while the most significant bit corresponds to drive P. The R/O bit is set either by an explicit call to function 28, or by the automatic software mechanisms within CP/M which detect changed disks.

```

*****
*
* FUNCTION 30: SET FILE ATTRIBUTES *
*
*****
* Entry Parameters: *
* Register C: 1EH *
* Registers DE: FCB Address *
*
* Returned Value: *
* Register A: Directory Code *
*****

```

The Set File Attributes function allows programmatic manipulation of permanent indicators attached to files. In particular, the R/O and System attributes (t1' and t2') can be set or reset. The DE pair addresses an unambiguous file name with the appropriate attributes set or reset. Function 30 searches for a match, and changes the matched directory entry to contain the selected indicators. Indicators f1' through f4' are not presently used, but may be useful for applications programs, since they are not involved in the matching process during file open and close operations. Indicators f5' through f8' and t3' are reserved for future system expansion.

```

*****
*
* FUNCTION 31: GET ADDR(DISK PARMS) *
*
*****
* Entry Parameters: *
*   Register C: 1FH *
*
* Returned Value: *
*   Registers HL: DPB Address *
*****

```

The address of the BIOS resident disk parameter block is returned in HL as a result of this function call. This address can be used for either of two purposes. First, the disk parameter values can be extracted for display and space computation purposes, or transient programs can dynamically change the values of current disk parameters when the disk environment changes, if required. Normally, application programs will not require this facility.

```

*****
*
* FUNCTION 32: SET/GET USER CODE *
*
*****
* Entry Parameters: *
*   Register C: 20H *
*   Register E: 0FFH (get) or *
*               User Code (set) *
*
* Returned Value: *
*   Register A: Current Code or *
*               (no value) *
*****

```

An application program can change or interrogate the currently active user number by calling function 32. If register E = 0FFH, then the value of the current user number is returned in register A, where the value is in the range 0 to 31. If register E is not 0FFH, then the current user number is changed to the value of E (modulo 32).

```

*****
*
* FUNCTION 33: READ RANDOM
*
*****
* Entry Parameters:
*   Register C: 21H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Return Code
*****

```

The Read Random function is similar to the sequential file read operation of previous releases, except that the read operation takes place at a particular record number, selected by the 24-bit value constructed from the three byte field following the FCB (byte positions r0 at 33, r1 at 34, and r2 at 35). Note that the sequence of 24 bits is stored with least significant byte first (r0), middle byte next (r1), and high byte last (r2). CP/M does not reference byte r2, except in computing the size of a file (function 35). Byte r2 must be zero, however, since a non-zero value indicates overflow past the end of file.

Thus, the r0,r1 byte pair is treated as a double-byte, or "word" value, which contains the record to read. This value ranges from 0 to 65535, providing access to any particular record of the 8 megabyte file. In order to process a file using random access, the base extent (extent 0) must first be opened. Although the base extent may or may not contain any allocated data, this ensures that the file is properly recorded in the directory, and is visible in DIR requests. The selected record number is then stored into the random record field (r0,r1), and the BDOS is called to read the record. Upon return from the call, register A either contains an error code, as listed below, or the value 00 indicating the operation was successful. In the latter case, the current DMA address contains the randomly accessed record. Note that contrary to the sequential read operation, the record number is not advanced. Thus, subsequent random read operations continue to read the same record.

Upon each random read operation, the logical extent and current record values are automatically set. Thus, the file can be sequentially read or written, starting from the current randomly accessed position. Note, however, that in this case, the last randomly read record will be re-read as you switch from random mode to sequential read, and the last record will be re-written as you switch to a sequential write operation. You can, of course, simply advance the random record position following each random read or write to obtain the effect of a sequential I/O operation.

Error codes returned in register A following a random read are listed below.

(All Information Contained Herein is Proprietary to Digital Research.)

- 01 reading unwritten data
- 02 (not returned in random mode)
- 03 cannot close current extent
- 04 seek to unwritten extent
- 05 (not returned in read mode)
- 06 seek past physical end of disk

Error code 01 and 04 occur when a random read operation accesses a data block which has not been previously written, or an extent which has not been created, which are equivalent conditions. Error 3 does not normally occur under proper system operation, but can be cleared by simply re-reading, or re-opening extent zero as long as the disk is not physically write protected. Error code 06 occurs whenever byte r2 is non-zero under the current 2.0 release. Normally, non-zero return codes can be treated as missing data, with zero return codes indicating operation complete.

```

*****
*
* FUNCTION 34: WRITE RANDOM
*
*****
* Entry Parameters:
*   Register C: 22H
*   Registers DE: FCB Address
*
* Returned Value:
*   Register A: Return Code
*****

```

The Write Random operation is initiated similar to the Read Random call, except that data is written to the disk from the current DMA address. Further, if the disk extent or data block which is the target of the write has not yet been allocated, the allocation is performed before the write operation continues. As in the Read Random operation, the random record number is not changed as a result of the write. The logical extent number and current record positions of the file control block are set to correspond to the random record which is being written. Again, sequential read or write operations can commence following a random write, with the notation that the currently addressed record is either read or rewritten again as the sequential operation begins. You can also simply advance the random record position following each write to get the effect of a sequential write operation. Note that in particular, reading or writing the last record of an extent in random mode does not cause an automatic extent switch as it does in sequential mode.

The error codes returned by a random write are identical to the random read operation with the addition of error code 05, which indicates that a new extent cannot be created due to directory overflow.


```

*****
*
* FUNCTION 35: COMPUTE FILE SIZE
*
*****
* Entry Parameters:
*   Register C: 23H
*   Registers DE: FCB Address
*
* Returned Value:
*   Random Record Field Set
*****

```

When computing the size of a file, the DE register pair addresses an FCB in random mode format (bytes r0, r1, and r2 are present). The FCB contains an unambiguous file name which is used in the directory scan. Upon return, the random record bytes contain the "virtual" file size which is, in effect, the record address of the record following the end of the file. If, following a call to function 35, the high record byte r2 is 01, then the file contains the maximum record count 65536. Otherwise, bytes r0 and r1 constitute a 16-bit value (r0 is the least significant byte, as before) which is the file size.

Data can be appended to the end of an existing file by simply calling function 35 to set the random record position to the end of file, then performing a sequence of random writes starting at the preset record address.

The virtual size of a file corresponds to the physical size when the file is written sequentially. If, instead, the file was created in random mode and "holes" exist in the allocation, then the file may in fact contain fewer records than the size indicates. If, for example, only the last record of an eight megabyte file is written in random mode (i.e., record number 65535), then the virtual size is 65536 records, although only one block of data is actually allocated.

```

*****
*
* FUNCTION 36: SET RANDOM RECORD
*
*****
* Entry Parameters:
*   Register C: 24H
*   Registers DE: FCB Address
*
* Returned Value:
*   Random Record Field Set
*****

```

The Set Random Record function causes the BDOS to automatically produce the random record position from a file which has been read or written sequentially to a particular point. The function can be useful in two ways.

First, it is often necessary to initially read and scan a sequential file to extract the positions of various "key" fields. As each key is encountered, function 36 is called to compute the random record position for the data corresponding to this key. If the data unit size is 128 bytes, the resulting record position is placed into a table with the key for later retrieval. After scanning the entire file and tabularizing the keys and their record numbers, you can move instantly to a particular keyed record by performing a random read using the corresponding random record number which was saved earlier. The scheme is easily generalized when variable record lengths are involved since the program need only store the buffer-relative byte position along with the key and record number in order to find the exact starting position of the keyed data at a later time.

A second use of function 36 occurs when switching from a sequential read or write over to random read or write. A file is sequentially accessed to a particular point in the file, function 36 is called which sets the record number, and subsequent random read and write operations continue from the selected point in the file.

3. A SAMPLE FILE-TO-FILE COPY PROGRAM.

The program shown below provides a relatively simple example of file operations. The program source file is created as COPY.ASM using the CP/M ED program and then assembled using ASM or MAC, resulting in a "HEX" file. The LOAD program is the used to produce a COPY.COM file which executes directly under the CCP. The program begins by setting the stack pointer to a local area, and then proceeds to move the second name from the default area at 006CH to a 33-byte file control block called DFCB. The DFCB is then prepared for file operations by clearing the current record field. At this point, the source and destination FCB's are ready for processing since the SFCB at 005CH is properly set-up by the CCP upon entry to the COPY program. That is, the first name is placed into the default fcb, with the proper fields zeroed, including the current record field at 007CH. The program continues by opening the source file, deleting any existing destination file, and then creating the destination file. If all this is successful, the program loops at the label COPY until each record has been read from the source file and placed into the destination file. Upon completion of the data transfer, the destination file is closed and the program returns to the CCP command level by jumping to BOOT.

```

;      sample file-to-file copy program
;
;      at the ccp level, the command
;
;      copy a:x.y b:u.v
;
;      copies the file named x.y from drive
;      a to a file named u.v on drive b.
;
0000 = boot      equ      0000h    ; system reboot
0005 = bdos      equ      0005h    ; bdos entry point
005c = fcbl      equ      005ch    ; first file name
005c = sfcbl     equ      fcbl     ; source fcb
006c = fcb2      equ      006ch    ; second file name
0080 = dbuff     equ      0080h    ; default buffer
0100 = tpa       equ      0100h    ; beginning of tpa
;
0009 = printf    equ      9        ; print buffer func#
000f = openf     equ      15       ; open file func#
0010 = closef    equ      16       ; close file func#
0013 = deletef   equ      19       ; delete file func#
0014 = readf     equ      20       ; sequential read
0015 = writef    equ      21       ; sequential write
0016 = makef     equ      22       ; make file func#
;
0100      org      tpa      ; beginning of tpa
0100 311b02  lxi      sp,stack; local stack
;
;      move second file name to dfcb
0103 0e10    mvi      c,16      ; half an fcb
```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0105 116c00      lxi      d,fc b2 ; source of move
0108 21da01      lxi      h,dfcb ; destination fcb
010b 1a          mfc b:  ldax   d      ; source fcb
010c 13          inx      d      ; ready next
010d 77          mov      m,a     ; dest fcb
010e 23          inx      h      ; ready next
010f 0d          dcr      c      ; count 16...0
0110 c20b01      jnz      mfc b  ; loop 16 times
;
;
; name has been moved, zero cr
0113 af          xra      a      ; a = 00h
0114 32fa01      sta      dfcbcr ; current rec = 0
;
;
; source and destination fcb's ready
;
0117 115c00      lxi      d,sfcb ; source file
011a cd6901      call     open   ; error if 255
011d 118701      lxi      d,nofile; ready message
0120 3c          inr      a      ; 255 becomes 0
0121 cc6101      cz       finis  ; done if no file
;
;
; source file open, prep destination
0124 11da01      lxi      d,dfcb ; destination
0127 cd7301      call     delete ; remove if present
;
;
012a 11da01      lxi      d,dfcb ; destination
012d cd8201      call     make   ; create the file
0130 119601      lxi      d,nodir ; ready message
0133 3c          inr      a      ; 255 becomes 0
0134 cc6101      cz       finis  ; done if no dir space
;
;
; source file open, dest file open
;
; copy until end of file on source
;
0137 115c00      copy:  lxi      d,sfcb ; source
013a cd7801      call     read   ; read next record
013d b7          ora      a      ; end of file?
013e c25101      jnz      eofile ; skip write if so
;
;
; not end of file, write the record
0141 11da01      lxi      d,dfcb ; destination
0144 cd7d01      call     write  ; write record
0147 11a901      lxi      d,space ; ready message
014a b7          ora      a      ; 00 if write ok
014b c46101      cnz      finis  ; end if so
014e c33701      jmp      copy   ; loop until eof
;
; eofile: ; end of file, close destination
0151 11da01      lxi      d,dfcb ; destination
0154 cd6e01      call     close  ; 255 if error
0157 21bb01      lxi      h,wrprot; ready message
015a 3c          inr      a      ; 255 becomes 00
015b cc6101      cz       finis  ; shouldn't happen
;
; copy operation complete, end

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

015e 1lcc01      lxi      d,normal; ready message
;
;finis:  ; write message given by de, reboot
0161 0e09      mvi      c,printf
0163 cd0500     call     bdos      ; write message
0166 c30000     jmp      boot      ; reboot system
;
;      system interface subroutines
;      (all return directly from bdos)
;
0169 0e0f      open:    mvi      c,openf
016b c30500     jmp      bdos
;
016e 0e10      close:   mvi      c,closef
0170 c30500     jmp      bdos
;
0173 0e13      delete:  mvi      c,deletef
0175 c30500     jmp      bdos
;
0178 0e14      read:    mvi      c,readf
017a c30500     jmp      bdos
;
017d 0e15      write:   mvi      c,writef
017f c30500     jmp      bdos
;
0182 0e16      make:    mvi      c,makef
0184 c30500     jmp      bdos
;
;      console messages
0187 6e6f20fno: db      'no source file$'
0196 6e6f209no: db      'no directory space$'
01a9 6f7574fspa: db      'out of data space$'
01bb 7772695wrp: db      'write protected?$'
01cc 636f700nor: db      'copy complete$'
;
;      data areas
01da          dfcb:     ds       33      ; destination fcb
01fa =        dfcbcr    equ      dfcb+32 ; current record
;
01fb          ds       32      ; 16 level stack
stack:
021b          end

```

Note that there are several simplifications in this particular program. First, there are no checks for invalid file names which could, for example, contain ambiguous references. This situation could be detected by scanning the 32 byte default area starting at location 005CH for ASCII question marks. A check should also be made to ensure that the file names have, in fact, been included (check locations 005DH and 006DH for non-blank ASCII characters). Finally, a check should be made to ensure that the source and destination file names are different. A speed improvement could be made by buffering more data on each read operation. One could, for example, determine

(All Information Contained Herein is Proprietary to Digital Research.)

the size of memory by fetching FBASE from location 0006H and use the entire remaining portion of memory for a data buffer. In this case, the programmer simply resets the DMA address to the next successive 128 byte area before each read. Upon writing to the destination file, the DMA address is reset to the beginning of the buffer and incremented by 128 bytes to the end as each record is transferred to the destination file.

4. A SAMPLE FILE DUMP UTILITY.

The file dump program shown below is slightly more complex than the simple copy program given in the previous section. The dump program reads an input file, specified in the CCP command line, and displays the content of each record in hexadecimal format at the console. Note that the dump program saves the CCP's stack upon entry, resets the stack to a local area, and restores the CCP's stack before returning directly to the CCP. Thus, the dump program does not perform a warm start at the end of processing.

```

; DUMP program reads input file and displays hex data
;
0100          org      100h
0005 =      bdos      equ      0005h      ;dos entry point
0001 =      cons      equ      1          ;read console
0002 =      typef     equ      2          ;type function
0009 =      printf    equ      9          ;buffer print entry
000b =      brkf      equ      11         ;break key function (true if char
000f =      openf     equ      15         ;file open
0014 =      readf     equ      20         ;read function

;
005c =      fcb       equ      5ch        ;file control block address
0080 =      buff      equ      80h        ;input disk buffer address
;
;      non graphic characters
000d =      cr        equ      0dh        ;carriage return
000a =      lf        equ      0ah        ;line feed
;
;      file control block definitions
005c =      fcdbn     equ      fcb+0      ;disk name
005d =      fcdbfn    equ      fcb+1      ;file name
0065 =      fcdbft    equ      fcb+9      ;disk file type (3 characters)
0068 =      fcdbrl    equ      fcb+12     ;file's current reel number
006b =      fcdbrc    equ      fcb+15     ;file's record count (0 to 128)
007c =      fcdbcr    equ      fcb+32     ;current (next) record number (0
007d =      fcdbln    equ      fcb+33     ;fcb length
;
;      set up stack
0100 210000    lxi      h,0
0103 39        dad      sp
;
;      entry stack pointer in hl from the ccp
0104 221502    shld     oldsp
;
;      set sp to local stack area (restored at finis)
0107 315702    lxi      sp,stktp
;
;      read and print successive buffers
010a cdcl01    call     setup      ;set up input file
010d feff      cpi      255        ;255 if file not present
010f c21b01    jnz      openok     ;skip if open is ok
;
;      file not there, give error message and return
0112 11f301    lxi      d,opnmsg
0115 cd9c01    call     err
0118 c35101    jmp      finis      ;to return
;

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

openok: ;open operation ok, set buffer index to end
011b 3e80      mvi      a,80h
011d 321302    sta      ibp      ;set buffer pointer to 80h
;             hl contains next address to print
0120 210000    lxi      h,0      ;start with 0000
;
gloop:
0123 e5        push     h          ;save line position
0124 cda201    call     gnb
0127 e1        pop      h          ;recall line position
0128 da5101    jc       finis      ;carry set by gnb if end file
012b 47        mov      b,a
;             print hex values
;             check for line fold
012c 7d        mov      a,l
012d e60f      ani      0fh        ;check low 4 bits
012f c24401    jnz      nonum
;             print line number
0132 cd7201    call     crlf
;
;             check for break key
0135 cd5901    call     break
;             accum lsb = 1 if character ready
0138 0f        rrc
0139 da5101    jc       finis      ;don't print any more
;
013c 7c        mov      a,h
013d cd8f01    call     phex
0140 7d        mov      a,l
0141 cd8f01    call     phex
nonum:
0144 23        inx      h          ;to next line number
0145 3e20      mvi      a,' '
0147 cd6501    call     pchar
014a 78        mov      a,b
014b cd8f01    call     phex
014e c32301    jmp      gloop
;
finis:
;             end of dump, return to ccp
;             (note that a jmp to 0000h reboots)
0151 cd7201    call     crlf
0154 2a1502    lhd      oldsp
0157 f9        sphl
;             stack pointer contains ccp's stack location
0158 c9        ret              ;to the ccp
;
;
;             subroutines
;
break: ;check break key (actually any key will do)
0159 e5d5c5    push     h! push d! push b; environment saved
015c 0e0b      mvi      c,brkf
015e cd0500    call     bdos
0161 c1d1e1    pop      b! pop d! pop h; environment restored

```

```

0164 c9          ret
;
; pchar: ;print a character
0165 e5d5c5      push h! push d! push b; saved
0168 0e02        mvi      c,typef
016a 5f          mov      e,a
016b cd0500      call     bdos
016e c1dle1      pop b! pop d! pop h; restored
0171 c9          ret
;
; crlf:
0172 3e0d        mvi      a,cr
0174 cd6501      call     pchar
0177 3e0a        mvi      a,lf
0179 cd6501      call     pchar
017c c9          ret
;
;
; pnib: ;print nibble in reg a
017d e60f        ani      0fh      ;low 4 bits
017f fe0a        cpi      10
0181 d28901      jnc      pl0
;               less than or equal to 9
0184 c630        adi      '0'
0186 c38b01      jmp      prn
;
;               greater or equal to 10
0189 c637        pl0:     adi      'a' - 10
018b cd6501      prn:     call     pchar
018e c9          ret
;
; phex: ;print hex char in reg a
018f f5          push     psw
0190 0f          rrc
0191 0f          rrc
0192 0f          rrc
0193 0f          rrc
0194 cd7d01      call     pnib      ;print nibble
0197 f1          pop      psw
0198 cd7d01      call     pnib
019b c9          ret
;
; err: ;print error message
;       d,e addresses message ending with "$"
019c 0e09        mvi      c,printf      ;print buffer function
019e cd0500      call     bdos
01a1 c9          ret
;
;
; gnb: ;get next byte
01a2 3a1302      lda      ibp
01a5 fe80        cpi      80h
01a7 c2b301      jnz      g0
;               read another buffer
;

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

;
01aa cdce01      call    diskr
01ad b7          ora     a      ;zero value if read ok
01ae cab301      jz      g0      ;for another byte
;              end of data, return with carry set for eof
01b1 37          stc
01b2 c9          ret
;
g0:              ;read the byte at buff+reg a
01b3 5f          mov     e,a      ;ls byte of buffer index
01b4 1600        mvi     d,0      ;double precision index to de
01b6 3c          inr     a      ;index=index+1
01b7 321302      sta     ibp      ;back to memory
;              pointer is incremented
;              save the current file address
01ba 218000      lxi     h,buff
01bd 19          dad     d
;              absolute character address is in hl
01be 7e          mov     a,m
;              byte is in the accumulator
01bf b7          ora     a      ;reset carry bit
01c0 c9          ret
;
setup:           ;set up file
;              open the file for input
01c1 af          xra     a      ;zero to accum
01c2 327c00      sta     fcbcr    ;clear current record
;
01c5 115c00      lxi     d,fcb
01c8 0e0f        mvi     c,openf
01ca cd0500      call    bdos
;              255 in accum if open error
01cd c9          ret
;
diskr:           ;read disk file record
01ce e5d5c5      push h! push d! push b
01d1 115c00      lxi     d,fcb
01d4 0e14        mvi     c,readf
01d6 cd0500      call    bdos
01d9 c1d1e1      pop b! pop d! pop h
01dc c9          ret
;
;              fixed message area
01dd 46494c0signon: db      'file dump version 2.0$'
01f3 0d0a4e0opnmsg: db      cr,lf,'no input file present on disk$'
;
;              variable area
0213            ibp:      ds      2      ;input buffer pointer
0215            oldsp:    ds      2      ;entry sp value from ccp
;
;              stack area
0217            stktop:   ds      64      ;reserve 32 level stack
;
0257            end

```

(All Information Contained Herein is Proprietary to Digital Research.)

5. A SAMPLE RANDOM ACCESS PROGRAM.

This manual is concluded with a rather extensive, but complete example of random access operation. The program listed below performs the simple function of reading or writing random records upon command from the terminal. Given that the program has been created, assembled, and placed into a file labelled RANDOM.COM, the CCP level command:

RANDOM X.DAT

starts the test program. The program looks for a file by the name X.DAT (in this particular case) and, if found, proceeds to prompt the console for input. If not found, the file is created before the prompt is given. Each prompt takes the form

next command?

and is followed by operator input, terminated by a carriage return. The input commands take the form

nW nR Q

where n is an integer value in the range 0 to 65535, and W, R, and Q are simple command characters corresponding to random write, random read, and quit processing, respectively. If the W command is issued, the RANDOM program issues the prompt

type data:

The operator then responds by typing up to 127 characters, followed by a carriage return. RANDOM then writes the character string into the X.DAT file at record n. If the R command is issued, RANDOM reads record number n and displays the string value at the console. If the Q command is issued, the X.DAT file is closed, and the program returns to the console command processor. In the interest of brevity, the only error message is

error, try again

The program begins with an initialization section where the input file is opened or created, followed by a continuous loop at the label "ready" where the individual commands are interpreted. The default file control block at 005CH and the default buffer at 0080H are used in all disk operations. The utility subroutines then follow, which contain the principal input line processor, called "readc." This particular program shows the elements of random access processing, and can be used as the basis for further program development.

```

;*****
;*
;* sample random access program for cp/m 2.0
;*
;*****
0100          org      100h      ;base of tpa
;
0000 = reboot equ      0000h    ;system reboot
0005 = bdos   equ      0005h    ;bdos entry point
;
0001 = coninp equ      1        ;console input function
0002 = conout equ      2        ;console output function
0009 = pstring equ      9        ;print string until '$'
000a = rstring equ     10        ;read console buffer
000c = version equ     12        ;return version number
000f = openf  equ     15        ;file open function
0010 = closef equ     16        ;close function
0016 = makef  equ     22        ;make file function
0021 = readr  equ     33        ;read random
0022 = writr  equ     34        ;write random
;
005c = fcb    equ     005ch     ;default file control block
007d = ranrec equ     fcb+33    ;random record position
007f = ranovf equ     fcb+35    ;high order (overflow) byte
0080 = buff   equ     0080h     ;buffer address
;
000d = cr     equ      0dh      ;carriage return
000a = lf     equ      0ah      ;line feed
;
;*****
;*
;* load SP, set-up file for random access
;*
;*****
0100 31bc0    lxi      sp,stack
;
;          version 2.0?
0103 0e0c    mvi      c,version
0105 cd050    call     bdos
0108 fe20    cpi      20h      ;version 2.0 or better?
010a d2160    jnc      versok
;          bad version, message and go back
010d 111b0    lxi      d,badver
0110 cdda0    call     print
0113 c3000    jmp      reboot
;
versok:
;          correct version for random access
0116 0e0f    mvi      c,openf ;open default fcb
0118 115c0    lxi      d,fcbl
011b cd050    call     bdos
011e 3c      inr      a        ;err 255 becomes zero
011f c2370    jnz      ready
;
;          cannot open file, so create it

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

0122 0e16      mvi      c,makef
0124 115c0     lxi      d,fcf
0127 cd050     call     bdos
012a 3c        inr      a          ;err 255 becomes zero
012b c2370     jnz      ready

;
;      cannot create file, directory full
012e 113a0     lxi      d,nospace
0131 cdda0     call     print
0134 c3000     jmp      reboot ;back to ccp

;
;*****
;*
;* loop back to "ready" after each command
;*
;*****
;
ready:
;      file is ready for processing
;
0137 cde50     call     readcom ;read next command
013a 227d0     shld     ranrec ;store input record#
013d 217f0     lxi      h,ranovf
0140 3600      mvi      m,0      ;clear high byte if set
0142 fe51      cpi      'Q'      ;quit?
0144 c2560     jnz      notq

;
;      quit processing, close file
0147 0e10      mvi      c,closef
0149 115c0     lxi      d,fcf
014c cd050     call     bdos
014f 3c        inr      a          ;err 255 becomes 0
0150 cab90     jz       error    ;error message, retry
0153 c3000     jmp      reboot ;back to ccp

;
;*****
;*
;* end of quit command, process write
;*
;*****
;
notq:
;      not the quit command, random write?
0156 fe57      cpi      'W'
0158 c2890     jnz      notw

;
;      this is a random write, fill buffer until cr
015b 114d0     lxi      d,datmsg
015e cdda0     call     print    ;data prompt
0161 0e7f      mvi      c,127   ;up to 127 characters
0163 21800     lxi      h,buff   ;destination
rloop: ;read next character to buff
0166 c5        push     b        ;save counter
0167 e5        push     h        ;next destination
0168 cdc20     call     getchr   ;character to a
016b e1        pop      h        ;restore counter

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

016c cl      pop      b      ;restore next to fill
016d fe0d    cpi      cr      ;end of line?
016f ca780   jz       erloop
;           not end, store character
0172 77      mov      m,a
0173 23      inx      h      ;next to fill
0174 0d      dcr      c      ;counter goes down
0175 c2660   jnz      rloop  ;end of buffer?
erloop:
;           end of read loop, store 00
0178 3600    mvi      m,0
;
;           write the record to selected record number
017a 0e22    mvi      c,writer
017c 115c0   lxi      d,fcbl
017f cd050   call     bdos
0182 b7      ora      a      ;error code zero?
0183 c2b90   jnz      error  ;message if not
0186 c3370   jmp      ready  ;for another record
;
;*****
;*
;* end of write command, process read
;*
;*****
notw:
;           not a write command, read record?
0189 fe52    cpi      'R'
018b c2b90   jnz      error  ;skip if not
;
;           read random record
018e 0e21    mvi      c,readr
0190 115c0   lxi      d,fcbl
0193 cd050   call     bdos
0196 b7      ora      a      ;return code 00?
0197 c2b90   jnz      error
;
;           read was successful, write to console
019a cdcf0   call     crlf    ;new line
019d 0e80    mvi      c,128  ;max 128 characters
019f 21800   lxi      h,buff  ;next to get
wloop:
01a2 7e      mov      a,m    ;next character
01a3 23      inx      h      ;next to get
01a4 e67f    ani      7fh    ;mask parity
01a6 ca370   jz       ready  ;for another command if 00
01a9 c5      push     b      ;save counter
01aa e5      push     h      ;save next to get
01ab fe20    cpi      ' '    ;graphic?
01ad d4c80   cnc      putchar ;skip output if not
01b0 e1      pop      h
01b1 c1      pop      b
01b2 0d      dcr      c      ;count=count-1
01b3 c2a20   jnz      wloop
01b6 c3370   jmp      ready

```

(All Information Contained Herein is Proprietary to Digital Research.)

```

;
;*****
;
; * end of read command, all errors end-up here *
; *
;*****
;
error:
01b9 11590      lxi      d,errmsg
01bc cdda0      call     print
01bf c3370      jmp      ready
;
;*****
;
; * utility subroutines for console i/o *
; *
;*****
getchr:
;read next console character to a
01c2 0e01      mvi      c,coninp
01c4 cd050      call     bdos
01c7 c9        ret
;
putchr:
;write character from a to console
01c8 0e02      mvi      c,conout
01ca 5f        mov      e,a      ;character to send
01cb cd050      call     bdos     ;send character
01ce c9        ret
;
crlf:
;send carriage return line feed
01cf 3e0d      mvi      a,cr      ;carriage return
01d1 cdc80      call     putchr
01d4 3e0a      mvi      a,lf      ;line feed
01d6 cdc80      call     putchr
01d9 c9        ret
;
print:
;print the buffer addressed by de until $
01da d5        push     d
01db cdcf0      call     crlf
01de d1        pop      d      ;new line
01df 0e09      mvi      c,pstring
01e1 cd050      call     bdos     ;print the string
01e4 c9        ret
;
readcom:
;read the next command line to the conbuf
01e5 116b0      lxi      d,prompt
01e8 cdda0      call     print    ;command?
01eb 0e0a      mvi      c,rstring
01ed 117a0      lxi      d,conbuf
01f0 cd050      call     bdos     ;read command line
;
command line is present, scan it

```

(All Information Contained Herein is Proprietary to Digital Research.)


```

01f3 21000      lxi      h,0      ;start with 0000
01f6 117c0      lxi      d,conlin;command line
01f9 1a         readc: ldax    d      ;next command character
01fa 13         inx      d      ;to next command position
01fb b7         ora      a      ;cannot be end of command
01fc c8         rz
;              not zero, numeric?
01fd d630      sui      '0'
01ff fe0a      cpi      10      ;carry if numeric
0201 d2130      jnc      endrd
;              add-in next digit
0204 29         dad      h      ;*2
0205 4d         mov      c,l
0206 44         mov      b,h      ;bc = value * 2
0207 29         dad      h      ;*4
0208 29         dad      h      ;*8
0209 09         dad      b      ;*2 + *8 = *10
020a 85         add      l      ;+digit
020b 6f         mov      l,a
020c d2f90      jnc      readc   ;for another char
020f 24         inr      h      ;overflow
0210 c3f90      jmp      readc   ;for another char
endrd:
;              end of read, restore value in a
0213 c630      adi      '0'      ;command
0215 fe61      cpi      'a'      ;translate case?
0217 d8         rc
;              lower case, mask lower case bits
0218 e65f      ani      101$1111b
021a c9         ret
;
;*****
;*
;* string data area for console messages
;*
;*****
badver:
021b 536f79     db      'sorry, you need cp/m version 2$'
nospace:
023a 4e6f29     db      'no directory space$'
datmsg:
024d 547970     db      'type data: $'
errmsg:
0259 457272     db      'error, try again.$'
prompt:
026b 4e6570     db      'next command? $'
;

```

```

;*****
;*
;* fixed and variable data area
;*
;*****
027a 21  conbuf: db      conlen  ;length of console buffer
027b      consiz: ds      1      ;resulting size after read
027c      conlin: ds     32      ;length 32 buffer
0021 =   conlen equ     $-consiz
;
029c      ds      32      ;16 level stack
stack:
02bc      end

```

Again, major improvements could be made to this particular program to enhance its operation. In fact, with some work, this program could evolve into a simple data base management system. One could, for example, assume a standard record size of 128 bytes, consisting of arbitrary fields within the record. A program, called GETKEY, could be developed which first reads a sequential file and extracts a specific field defined by the operator. For example, the command

```
GETKEY NAMES.DAT  LASTNAME 10 20
```

would cause GETKEY to read the data base file NAMES.DAT and extract the "LASTNAME" field from each record, starting at position 10 and ending at character 20. GETKEY builds a table in memory consisting of each particular LASTNAME field, along with its 16-bit record number location within the file. The GETKEY program then sorts this list, and writes a new file, called LASTNAME.KEY, which is an alphabetical list of LASTNAME fields with their corresponding record numbers. (This list is called an "inverted index" in information retrieval parlance.)

Rename the program shown above as QUERY, and massage it a bit so that it reads a sorted key file into memory. The command line might appear as:

```
QUERY NAMES.DAT LASTNAME.KEY
```

Instead of reading a number, the QUERY program reads an alphanumeric string which is a particular key to find in the NAMES.DAT data base. Since the LASTNAME.KEY list is sorted, you can find a particular entry quite rapidly by performing a "binary search," similar to looking up a name in the telephone book. That is, starting at both ends of the list, you examine the entry halfway in between and, if not matched, split either the upper half or the lower half for the next search. You'll quickly reach the item you're looking for (in $\log_2(n)$ steps) where you'll find the corresponding record number. Fetch and display this record at the console, just as we have done in the program shown above.

(All Information Contained Herein is Proprietary to Digital Research.)

At this point you're just getting started. With a little more work, you can allow a fixed grouping size which differs from the 128 byte record shown above. This is accomplished by keeping track of the record number as well as the byte offset within the record. Knowing the group size, you randomly access the record containing the proper group, offset to the beginning of the group within the record read sequentially until the group size has been exhausted.

Finally, you can improve QUERY considerably by allowing boolean expressions which compute the set of records which satisfy several relationships, such as a LASTNAME between HARDY and LAUREL, and an AGE less than 45. Display all the records which fit this description. Finally, if your lists are getting too big to fit into memory, randomly access your key files from the disk as well. One note of consolation after all this work: if you make it through the project, you'll have no more need for this manual!

6. SYSTEM FUNCTION SUMMARY.

FUNC	FUNCTION NAME	INPUT PARAMETERS	OUTPUT RESULTS
0	System Reset	none	none
1	Console Input	none	A = char
2	Console Output	E = char	none
3	Reader Input	none	A = char
4	Punch Output	E = char	none
5	List Output	E = char	none
6	Direct Console I/O	see def	see def
7	Get I/O Byte	none	A = IOBYTE
8	Set I/O Byte	E = IOBYTE	none
9	Print String	DE = .Buffer	none
10	Read Console Buffer	DE = .Buffer	see def
11	Get Console Status	none	A = 00/FF
12	Return Version Number	none	HL= Version*
13	Reset Disk System	none	see def
14	Select Disk	E = Disk Number	see def
15	Open File	DE = .FCB	A = Dir Code
16	Close File	DE = .FCB	A = Dir Code
17	Search for First	DE = .FCB	A = Dir Code
18	Search for Next	none	A = Dir Code
19	Delete File	DE = .FCB	A = Dir Code
20	Read Sequential	DE = .FCB	A = Err Code
21	Write Sequential	DE = .FCB	A = Err Code
22	Make File	DE = .FCB	A = Dir Code
23	Rename File	DE = .FCB	A = Dir Code
24	Return Login Vector	none	HL= Login Vect*
25	Return Current Disk	none	A = Cur Disk#
26	Set DMA Address	DE = .DMA	none
27	Get Addr(Alloc)	none	HL= .Alloc
28	Write Protect Disk	none	see def
29	Get R/O Vector	none	HL= R/O Vect*
30	Set File Attributes	DE = .FCB	see def
31	Get Addr(disk parms)	none	HL= .DPB
32	Set/Get User Code	see def	see def
33	Read Random	DE = .FCB	A = Err Code
34	Write Random	DE = .FCB	A = Err Code
35	Compute File Size	DE = .FCB	r0, r1, r2
36	Set Random Record	DE = .FCB	r0, r1, r2

* Note that A = L, and B = H upon return

(All Information Contained Herein is Proprietary to Digital Research.)

MACRO - 80

Assembler

Reference Manual

(c) Microsoft - 1981

All Rights Reserved Worldwide

10-642-1045

Information in this document is subject to change without notice and does not represent a commitment on the part of Microsoft. The software described in this document is furnished under a license agreement or non-disclosure agreement. The software may be used or copied only in accordance with the terms of the agreement.

(C) Microsoft, 1979

NOTE: Chapters 2 and 3 of this manual have been deleted as the associated software is not provided as part of the Xerox software.

CP/M is a registered trade mark of Digital Research

ADDENDA TO: Utility Software Manual
MACRO-80 Assembler Reference Manual
XMACRO-86 Assembler Reference Manual

The following features were added or modified in release 3.4.

Add to Section 2.2.2 Switches

<u>Switch</u>	<u>Action</u>
---------------	---------------

/M	Initialize Block Data Areas. If the programmer wants the area that is defined by the DS (Define Space) pseudo-op initialized to zeros, then the programmer should use the /M switch in the command line. Otherwise, the space is not guaranteed to contain zeros. That is, DS does not automatically initialize the space to zeros.
----	--

/X	The presence or absence of /X in the command line sets the initial current mode and the initial value of the default for listing or suppressing lines in false conditional blocks. /X sets the current mode and initial value of default to not-to-list. No /X sets current mode and initial value of default to list. Current mode determines whether false conditionals will be listed or suppressed. The initial value of the default is used with the .TFCOND pseudo-op so that .TFCOND is independent of .SFCOND and .LFCOND. If the program contains .SFCOND or .LFCOND, /X has no effect after .SFCOND or .LFCOND is encountered until a .TFCOND is encountered in the file. So /X has an effect only when used with a file that contains no conditional listing pseudo-ops or when used with .TFCOND.
----	---

The following chart illustrates the effects of the three pseudo-ops when encountered under /X and under no /X. See the addition to Section 2.6.27 below for a full description of the three conditional listing pseudo-ops.

<u>PSEUDO-OP</u>	<u>NO /X</u>	<u>/X</u>
(none)	ON	OFF
.	.	.
.SFCOND	OFF	OFF
.	.	.
.LFCOND	ON	ON
.	.	.
.TFCOND	OFF	ON
.	.	.
.TFCOND	ON	OFF
.	.	.
.SFCOND	OFF	OFF
.	.	.
.TFCOND	OFF	ON
.TFCOND	ON	OFF
.	.	.
.TFCOND	OFF	ON

Add to Section 2.6.26 Conditional Pseudo Operations

IFIDN <arg1>,<arg2> True if the string <arg1> is
 IDeNtical to the string <arg2>.
 The angle brackets around <arg1>
 and <arg2> are required.

IFDIF <arg1>,<arg2> True if the string <arg1> is
 DIFFerent from the string <arg2>.
 The angle brackets around <arg1>
 and <arg2> are required.

Add to Section 2.6.27 Listing Control Pseudo Operations

There are now five listing control pseudo-ops. Output to the listing file can be controlled by the following pseudo-ops:

.LIST, .XLIST, .SFCOND, .LFCOND, .TFCOND

The three new pseudo-ops control the listing of conditional pseudo-op blocks which evaluate as false. These pseudo-ops give the programmer control over four cases.

1. Normally list false conditionals
For this case, the programmer simply allows the default mode to control the listing. The default mode is list false conditionals. If the programmer decides to suppress false conditionals, the /X switch can be issued in the command line instead of editing the source file.
2. Normally suppress false conditionals
For this case, the programmer issues the .TFCOND pseudo-op in the program file. .TFCOND reverses (toggles) the default, causing false conditionals to be suppressed. If the programmer decides to list false conditionals, the /X switch can be issued in the command line instead of editing the source file.
3. Always suppress/list false conditionals
For these cases, the programmer issues either the .SFCOND pseudo-op to suppress false conditionals, or the .LFCOND pseudo-op to list all false conditionals.
4. Suppress/list some false conditionals
For this case, the programmer has decided for most false conditionals whether to list or suppress, but for some false conditionals the programmer has not yet decided. For the false conditionals decided about, use .SFCOND or .LFCOND. For those not yet decided, use .TFCOND. .TFCOND sets the current and default settings to the opposite of the default. Initially, the default is set by giving /X or no /X in the command line. Two subcases exist:
 1. The programmer wants some false conditionals not to list unless /X is given. The programmer uses the .SFCOND and .LFCOND pseudo-ops to control which areas always suppress or list false conditionals. To selectively suppress some false conditionals, the programmer issues .TFCOND at the beginning of the conditional block and again at the end of the conditional block. (NOTE: The second .TFCOND is should be so that the default setting will be the same as the initial setting. Leaving the default equal

to the initial setting makes it easier to keep track of the default mode if there are many such areas.) If the conditional block evaluates as false, the lines will be suppressed. In this subcase, issuing the /X switch in the command line causes the conditional block affected by .TFCOND to list even if it evaluates as false.

2. The programmer wants some false conditionals to list unless /X is given. of the file. Two consecutive .TFCONDs places the conditional listing setting in initial state which is determined by the presence or absence of the /X switch (the first .TFCOND sets the default to not initial; the second to initial). The selected conditional block then responds to the /X switch: if a /X switch is issued in the command line, the conditional block is suppressed if false; if no /X switch is issued in the command line, the conditional block is listed even if false.

The programmer then must reissue the .SFCOND or .LFCOND conditional listing pseudo-op to restore the suppress or list mode. Simply issuing another .TFCOND will not restore the prior mode, but will toggle the default setting. Since in this subcase, the next area of code is supposed to list or suppress false conditionals always, the programmer must issue .SFCOND or .LFCOND.

The three conditional listing pseudo-ops are summarized below.

<u>PSEUDO-OP</u>	<u>DEFINITION</u>
.SFCOND	Suppresses the listing of conditional blocks that evaluate as false.
.LFCOND	Restores the listing of conditional blocks that evaluate as false.
.TFCOND	Toggles the current setting which controls the listing false conditionals. .TFCOND sets the current and default setting to not default. If a /X switch is given in the MACRO-80 run command line for a file which contains .TFCOND, /X reverses the effect of .TFCOND.

Add to Section 2.7.9 Special Macro Operators and Forms

% The percent sign is used only in a macro argument. % converts the expression that follows it (usually a symbol) to a number in the current radix. During macro expansion, the number derived from converting the expression is substituted for the dummy. Using the % special operator allows a macro call by value. (Usually, a macro call is a call by reference with the text of the macro argument substituting exactly for the dummy.)

The expression following the % must conform to the same rules as the DS (Define Space) pseudo-op. A valid expression returning a non-relocatable constant is required.

EXAMPLE:

Normally, LB, the argument to MAKLAB, would be substituted for Y, the argument to MACRO, as a string. The % causes LB to be converted to a non-relocatable constant which is then substituted for Y. Without the % special operator, the result of assembly would be 'Error LB' rather than 'Error 1', etc.

MAKLAB	MACRO	Y
ERR&Y:	DB	'Error &Y',0
	ENDM	
MAKERR	MACRO	X
LB	SET	0
	REPT	X
LB	SET	LB+1
	MAKLAB	%LB
	ENDM	
	ENDM	

When called by MAKERR 3, the assembler will generate:

ERR1:	DB	'Error 1',0
ERR2:	DB	'Error 2',0
ERR3:	DB	'Error 3',0

1944-1945

1946-1947

1948-1949

1950-1951

1952-1953

1954-1955

1956-1957

1958-1959

1960-1961

1962-1963

1964-1965

Microsoft

CONTENTS

CHAPTER 1 Introduction

CHAPTER 2 MACRO-80 Assembler

- 2.1 Running MACRO-80
- 2.2 Command Format
 - 2.2.1 Devices
 - 2.2.2 Switches
- 2.3 Format of MACRO-80 Source Files
 - 2.3.1 Statements
 - 2.3.2 Symbols
 - 2.3.3 Numeric Constants
 - 2.3.4 Strings
- 2.4 Expression Evaluation
 - 2.4.1 Arithmetic and Logical Operators
 - 2.4.2 Modes
 - 2.4.3 Externals
- 2.5 Opcodes as Operands
- 2.6 Pseudo Operations
 - 2.6.1 ASEG
 - 2.6.2 COMMON
 - 2.6.3 CSEG
 - 2.6.4 DB - Define Byte
 - 2.6.5 DC - Define Character
 - 2.6.6 DS - Define Space
 - 2.6.7 DSEG
 - 2.6.8 DW - Define Word
 - 2.6.9 END
 - 2.6.10 ENTRY/PUBLIC
 - 2.6.11 EQU
 - 2.6.12 EXT/EXTRN
 - 2.6.13 INCLUDE
 - 2.6.14 NAME
 - 2.6.15 ORG - Define Origin
 - 2.6.16 PAGE
 - 2.6.17 SET
 - 2.6.18 SUBTTL
 - 2.6.19 TITLE
 - 2.6.20 .COMMENT
 - 2.6.21 .PRINTX
 - 2.6.22 .RADIX
 - 2.6.23 .Z80
 - 2.6.24 .8080
 - 2.6.25 .REQUEST
 - 2.6.26 Conditional Pseudo Operations
 - 2.6.26.1 ELSE
 - 2.6.26.2 ENDIF
 - 2.6.27 Listing Control Pseudo Operations

- 2.6.28 Relocation Pseudo Operations
 - 2.6.28.1 ORG Pseudo-op
 - 2.6.28.2 LINK-80
- 2.6.29 Relocation Before Loading
- 2.7 Macros and Block Pseudo Operations
 - 2.7.1 Terms
 - 2.7.2 REPT-ENDM
 - 2.7.3 IRP-ENDM
 - 2.7.4 IRPC-ENDM
 - 2.7.5 MACRO
 - 2.7.6 ENDM
 - 2.7.7 EXITM
 - 2.7.8 LOCAL
 - 2.7.9 Special Macro Operators and Forms
- 2.8 Using Z80 Pseudo-ops
- 2.9 Sample Assembly
- 2.10 MACRO-80 Errors
- 2.11 Compatibility with Other Assemblers
- 2.12 Format of Listings
 - 2.12.1 Symbol Table Listing

CHAPTER 3 CREF-80 Cross Reference Facility

CHAPTER 4 LINK-80 Linking Loader

- 4.1 Running LINK-80
- 4.2 Command Format
 - 4.2.1 LINK-80 Switches
 - 4.2.2 Sample Link
- 4.3 Format of LINK Compatible Object Files
- 4.4 LINK-80 Error Messages
- 4.5 Program Break Information

CHAPTER 5 LIB-80 Library Manager

- 5.1 LIB-80 Commands
 - 5.1.1 Modules
- 5.2 LIB-80 Switches
- 5.3 LIB-80 Listings
- 5.4 Sample LIB Session
- 5.5 Summary of Switches and Syntax

APPENDIX A TEKDOS Operating System

- A.1 TEKDOS Command Files
- A.2 MACRO-80
- A.3 CREF-80
- A.4 LINK-80

CHAPTER 1

INTRODUCTION

MACRO-80 is a relocatable macro assembler for 8080 and Z80 microcomputer systems. It assembles 8080 or Z80 code on any 8080 or Z80 development system running the CP/M, ISIS-II, TRSDOS or TEKDOS operating system. The MACRO-80 package includes the MACRO-80 assembler, the LINK-80 linking loader, and the CREF-80 cross reference facility. CP/M versions also include the LIB-80 Library Manager. MACRO-80 resides in approximately 14K of memory and has an assembly rate of over 1000 lines per minute.

MACRO-80 incorporates almost all "big computer" assembler features without sacrificing speed or memory space. The assembler supports a complete, Intel standard macro facility, including IRP, IRPC, REPEAT, local variables and EXITM. Nesting of macros is limited only by memory. Code is assembled in relocatable modules that are manipulated with the flexible linking loader. Conditional assembly capability is enhanced by an expanded set of conditional pseudo operations that include testing of assembly pass, symbol definition, and parameters to macros. Conditionals may be nested up to 255 levels.

MACRO-80's linking loader provides a versatile array of loader capabilities, which are executed by means of easy command lines and switches. Any number of programs may be loaded with one command, relocatable modules may be loaded in user-specified locations, and external references between modules are resolved automatically by the loader. The loader also performs library searches for system subroutines and generates a load map of memory showing the locations of the main program and subroutines. The cross reference facility that is included in this package supplies a convenient alphabetic list of all program variable names, along with the line numbers where they are referenced and defined.

This manual is designed to serve as a reference guide to the MACRO-80 package. It defines, explains and gives examples of all the features in MACRO-80 in terms that should be understandable to anyone familiar with assembly language programming. It is not intended, however, to serve as instructional material and presumes the user has substantial knowledge of assembly language programming. The user should refer to instructional material available from a variety of sources for additional tutorial information.

CHAPTER 2

MACRO-80 ASSEMBLER

2.1 RUNNING MACRO-80

The command to run MACRO-80 is

M80

MACRO-80 returns the prompt "*", indicating it is ready to accept commands.

NOTE

If you are using the TEKDOS operating system, see Appendix A for proper command formats.

2.2 COMMAND FORMAT

A command to MACRO-80 consists of a string of filenames with optional switches. All filenames should follow the operating system's conventions for filenames and extensions. The default extensions supplied by Microsoft software are as follows:

<u>File</u>	<u>CP/M</u>	<u>ISIS-II</u>
Relocatable object file	REL	REL
Listing file	PRN	LST
MACRO-80 source file	MAC	MAC
FORTTRAN source file	FOR	FOR
COBOL source	COB	COB
Absolute file	COM	

A command to MACRO-80 conveys the name of the source file to be assembled, the names of the file(s) to be created, and which assembly options are desired. The format of a MACRO-80 command is:

objfile,lstfile=source file

Only the equal sign and the source file field are required to create a relocatable object file with the default (source) filename and the default extension REL.

Otherwise, an object file is created only if the objfile field is filled, and a listing file is created only if the lstfile field is filled.

To assemble the source file without producing an object file or listing file, place only a comma to the left of the equal sign. This is a handy procedure that lets you check for syntax errors before assembling to an object file.

Examples:

<code>*=TEST</code>	Assemble the source file TEST.MAC and place the object file in TEST.REL.
<code>*,=TEST</code>	Assemble the source file TEST.MAC without creating an object or listing file. Useful for error checking.
<code>TEST,TEST=TEST</code>	Assemble the source file TEST.MAC, placing the object file in TEST.REL and the listing file in TEST.PRN. (With ISIS-II, the listing file is TEST.LST.)
<code>*OBJECT=TEST</code>	Assemble the source file TEST.MAC and place the object file in OBJECT.REL.
<code>OBJECT,LIST=TEST</code>	Assemble the source file TEST.MAC, placing the object file in OBJECT.REL and the listing file in LIST.PRN. (With ISIS-II, the listing file is LIST.LST.)

MACRO-80 also supports command lines; that is, the invocation and command may be typed on the same line. For example:

`M80 ,=TEST`

2.2.1 Devices

Any field in the MACRO-80 command string can also specify a device name. The default device name with the CP/M operating system is the currently logged disk. The default device name with the ISIS-II operating system is disk drive 0. The command format is:

dev:objfile,dev:lstfile=dev:source file

The device names are as follows:

<u>Device</u>	<u>CP/M</u>	<u>ISIS-II</u>
Disk drives	A:, B:, C:,...	:F0:, :F1:, :F2:, ...
Line printer	LST:	LST:
Teletype or CRT	TTY:	TTY:
High speed reader	HSR	

Examples:

* ,TTY:=TEST	Assemble the source file TEST.MAC and list the program on the console. No object code is generated. Useful for error check.
*SMALL,TTY:=B:TEST	Assemble TEST.MAC (found on disk drive B), place the object file in SMALL.REL, and list the program on the console.

2.2.2 Switches

A switch is a letter that is appended to the command string, preceded by a slash. It specifies an optional task to be performed during assembly. More than one switch can be used, but each must be preceded by a slash. (With the TEKDOS operating system, switches are preceded by commas or spaces. See Appendix A.) All switches are optional. The available switches are:

<u>Switch</u>	<u>Action</u>
O	Octal listing
H	Hexadecimal listing (default)
R	Force generation of an object file
L	Force generation of a listing file
C	Force generation of a cross reference file

- Z Assemble 280 opcodes (default for 280 operating systems)
- I Assemble 8080 opcodes (default for 8080 operating systems)
- P Each /P allocates an extra 256 bytes of stack space for use during assembly. Use /P if stack overflow errors occur during assembly. Otherwise, not needed.
- M Initialize Block Data Areas. If the programmer wants the area that is defined by the DS (Define Space) pseudo-op initialized to zeros, then the programmer should use the /M switch in the command line. Otherwise, the space is not guaranteed to contain zeros. That is, DS does not automatically initialize the space to zeros.
- X Usually used to suppress the listing of false conditionals. The following paragraph describes the /X switch more completely but in very technical terms.

The presence or absence of /X in the command line sets the initial current mode and the initial value of the default for listing or suppressing lines in false conditional blocks. /X sets the current mode and initial value of default to not-to-list. No /X sets current mode and initial value of default to list. Current mode determines whether false conditionals will be listed or suppressed. The initial value of the default is used with the .TFCOND pseudo-op so that .TFCOND is independent of .SFCOND and .LFCOND. If the program contains .SFCOND or .LFCOND, /X has no effect after .SFCOND or .LFCOND is encountered until a .TFCOND is encountered in the file. So /X has an effect only when used with a file that contains no conditional listing pseudo-ops or when used with .TFCOND.

Examples:

`*=TEST/L` Assemble TEST.MAC, place the object file in TEST.REL and a listing file in TEST.PRN. (With ISIS-II, the listing file is TEST.LST.)

`*=TEST/L/O` Same as above, but listing file addresses will be in octal.

`*LAST=TEST/C` Assemble TEST.MAC, place the object file in LAST.REL and cross reference file in TEST.CRF. (See Chapter 3.)

2.3 FORMAT OF MACRO-80 SOURCE FILES

Input source lines of up to 132 characters in length are acceptable.

MACRO-80 preserves lower case letters in quoted strings and comments. All symbols, opcodes and pseudo-opcodes typed in lower case will be converted to upper case.

If the source file includes line numbers from an editor, each byte of the line number must have the high bit on. Line numbers from Microsoft's EDIT-80 Editor are acceptable.

2.3.1 Statements

Source files input to MACRO-80 consist of statements of the form:

```
[label[:]] [operator] [arguments] [;comment]
```

With the exception of the ISIS assembler \$ controls (see Section 2.11), it is not necessary that statements begin in column 1. Multiple blanks or tabs may be used to improve readability.

If a label is present, it is the first item in the statement and is immediately followed by a colon. If it is followed by two colons, it is declared as PUBLIC (see ENTRY/PUBLIC, Section 2.6.10). For example:

```
FOO:: RET
```

is equivalent to

```
PUBLIC FOO
FOO: RET
```

The next item after the label, or the first item on the line if no label is present, is an operator. An operator may be an 8086 mnemonic, pseudo-op, macro call or expression. The evaluation order is as follows:

1. Macro call
2. Mnemonic/Pseudo operation
3. Expression

Instead of flagging an expression as an error, the assembler treats it as if it were a DB statement (see Section 2.6.4).

The arguments following the operator will, of course, vary in form according to the operator.

A comment always begins with a semicolon and ends with a carriage return. A comment may be a line by itself or it may be appended to a line that contains a statement. Extended comments can be entered using the .COMMENT pseudo operation (see Section 2.6.20).

2.3.2 Symbols

MACRO-80 symbols may be of any length, however, only the first six characters are significant. The following characters are legal in a symbol:

A-Z 0-9 \$. ? @

With Microsoft's 8080/Z80/8086 assemblers, the underline character is also legal in a symbol. A symbol may not start with a digit. When a symbol is read, lower case is translated into upper case. If a symbol reference is followed by ## it is declared external (see also the EXT/EXTRN pseudo-op, Section 2.6.12).

2.3.3 Numeric Constants

The default base for numeric constants is decimal. This may be changed by the .RADIX pseudo-op (see Section 2.6.22). Any base from 2 (binary) to 16 (hexadecimal) may be selected. When the base is greater than 10, A-F are the digits following 9. If the first digit of the number is not numeric the number must be preceded by a zero.

Numbers are 16-bit unsigned quantities. A number is always evaluated in the current radix unless one of the following special notations is used:

nnnnB	Binary
nnnnD	Decimal
nnnnO	Octal
nnnnQ	Octal
nnnnH	Hexadecimal
X'nnnn'	Hexadecimal

Overflow of a number beyond two bytes is ignored and the result is the low order 16-bits.

A character constant is a string comprised of zero, one or two ASCII characters, delimited by quotation marks, and used in a non-simple expression. For example, in the statement

```
DB      'A' + 1
```

'A' is a character constant. But the statement

```
DB      'A'
```

uses 'A' as a string because it is in a simple expression. The rules for character constant delimiters are the same as for strings.

A character constant comprised of one character has as its value the ASCII value of that character. That is, the high order byte of the value is zero, and the low order byte is the ASCII value of the character. For example, the value of the constant 'A' is 41H.

A character constant comprised of two characters has as its value the ASCII value of the first character in the high order byte and the ASCII value of the second character in the low order byte. For example, the value of the character constant "AB" is 41H*256+42H.

2.3.4 Strings

A string is comprised of zero or more characters delimited by quotation marks. Either single or double quotes may be used as string delimiters. The delimiter quotes may be used as characters if they appear twice for every character occurrence desired. For example, the statement

```
DB      "I am ""great"" today"
```

stores the string

```
I am "great" today
```

If there are zero characters between the delimiters, the string is a null string.

2.4 EXPRESSION EVALUATION

2.4.1 Arithmetic And Logical Operators

The following operators are allowed in expressions. The operators are listed in order of precedence.

NUL

LOW, HIGH

*, /, MOD, SHR, SHL

Unary Minus

+, -

EQ, NE, LT, LE, GT, GE

NOT

AND

OR, XOR

Parentheses are used to change the order of precedence. During evaluation of an expression, as soon as a new operator is encountered that has precedence less than or equal to the last operator encountered, all operations up to the new operator are performed. That is, subexpressions involving operators of higher precedence are computed first.

All operators except +, -, *, / must be separated from their operands by at least one space.

The byte isolation operators (HIGH, LOW) isolate the high or low order 8 bits of an Absolute 16-bit value. If a relocatable value is supplied as an operand, HIGH and LOW will treat it as if it were relative to location zero.

2.4.2 Modes

All symbols used as operands in expressions are in one of the following modes: Absolute, Data Relative, Program (Code) Relative or COMMON. (See Section 2.6 for the ASEG, CSEG, DSEG and COMMON pseudo-ops.) Symbols assembled under the ASEG, CSEG (default), or DSEG pseudo-ops are in Absolute, Code Relative or Data Relative mode respectively.

The number of COMMON modes in a program is determined by the number of COMMON blocks that have been named with the COMMON pseudo-op. Two COMMON symbols are not in the same mode unless they are in the same COMMON block. In any operation other than addition or subtraction, the mode of both operands must be Absolute.

If the operation is addition, the following rules apply:

1. At least one of the operands must be Absolute.
2. Absolute + <mode> = <mode>

If the operation is subtraction, the following rules apply:

1. <mode> - Absolute = <mode>
2. <mode> - <mode> = Absolute
where the two <mode>s are the same.

Each intermediate step in the evaluation of an expression must conform to the above rules for modes, or an error will be generated. For example, if FOO, BAZ and ZAZ are three Program Relative symbols, the expression

FOO + BAZ - ZAZ

will generate an R error because the first step (FOO + BAZ) adds two relocatable values. (One of the values must be Absolute.) This problem can always be fixed by inserting parentheses. So that

FOO + (BAZ - ZAZ)

is legal because the first step (BAZ - ZAZ) generates an Absolute value that is then added to the Program Relative value, FOO.

2.4.3 Externals

Aside from its classification by mode, a symbol is either External or not External. (See EXT/EXTRN, Section 2.6.12.) An External value must be assembled into a two-byte field. (Single-byte Externals are not supported.) The following rules apply to the use of Externals in expressions:

1. Externals are legal only in addition and subtraction.
2. If an External symbol is used in an expression, the result of the expression is always External.
3. When the operation is addition, either operand (but not both) may be External.

4. When the operation is subtraction, only the first operand may be External.

2.5 OPCODES AS OPERANDS

8080 opcodes are valid one-byte operands. Note that only the first byte is a valid operand. For example:

```
MVI    A,(JMP)
ADI     (CPI)
MVI     B,(RNZ)
CPI     (INX H)
ACI     (LXI B)
MVI     C,MOV A,B
```

Errors will be generated if more than one byte is included in the operand -- such as (CPI 5), LXI B,LABEL1) or (JMP LABEL2).

Opcodes used as one-byte operands need not be enclosed in parentheses.

NOTE

Opcodes are not valid operands
in Z80 mode.

2.6 PSEUDO OPERATIONS

2.6.1 ASEG

ASEG

ASEG sets the location counter to an absolute segment of memory. The location of the absolute counter will be that of the last ASEG (default is 0), unless an ORG is done after the ASEG to change the location. The effect of ASEG is also achieved by using the code segment (CSEG) pseudo operation and the /P switch in LINK-80. See also Section 2.6.28

2.6.2 COMMON

COMMON /<block name>/

COMMON sets the location counter to the selected common block in memory. The location is always the beginning of the area so that compatibility with the FORTRAN COMMON statement is maintained. If <block name> is omitted or consists of spaces, it is considered to be blank common. See also Section 2.6.28.

2.6.3 CSEG

CSEG

CSEG sets the location counter to the code relative segment of memory. The location will be that of the last CSEG (default is 0), unless an ORG is done after the CSEG to change the location. CSEG is the default condition of the assembler (the INTEL assembler defaults to ASEG). See also Section 2.6.28.

2.6.4 DB - Define Byte

DB <exp>[,<exp>...]

DB <string>[<string>...]

The arguments to DB are either expressions or strings. DB stores the values of the expressions or the characters of the strings in successive memory locations beginning with the current location counter.

Expressions must evaluate to one byte. (If the high byte of the result is 0 or 255, no error is given; otherwise, an A error results.)

Strings of three or more characters may not be used in expressions (i.e., they must be immediately followed by a comma or the end of the line). The characters in a string are stored in the order of appearance, each as a one-byte value with the high order bit set to zero.

Example:

```
0000'   41 42           DB      'AB'
0002'   42              DB      'AB' AND 0FFH
0003'   41 42 43        DB      'ABC'
```

2.6.5 DC - Define Character

DC <string>

DC stores the characters in <string> in successive memory locations beginning with the current location counter. As with DB, characters are stored in order of appearance, each as a one-byte value with the high order bit set to zero. However, DC stores the last character of the string with the high order bit set to one. An error will result if the argument to DC is a null string.

2.6.6 DS - Define Space

DS <exp>

DS reserves an area of memory. The value of <exp> gives the number of bytes to be allocated. All names used in <exp> must be previously defined (i.e., all names known at that point on pass 1). Otherwise, a V error is generated during pass 1 and a U error may be generated during pass 2. If a U error is not generated during pass 2, a phase error will probably be generated because the DS generated no code on pass 1.

2.6.7 DSEG

DSEG

DSEG sets the location counter to the Data Relative segment of memory. The location of the data relative counter will be that of the last DSEG (default is 0), unless an ORG is

done after the DSEG to change the location. See also Section 2.6.28.

2.6.8 DW - Define Word

DW <exp>[,<exp>...]

DW stores the values of the expressions in successive memory locations beginning with the current location counter. Expressions are evaluated as 2-byte (word) values.

2.6.9 END

END [<exp>]

The END statement specifies the end of the program. If <exp> is present, it is the start address of the program. If <exp> is not present, then no start address is passed to LINK-80 for that program.

NOTE

If an assembly language program is the main program, a start address (label) must be specified. If not, LINK-80 will issue a "no start address" error. If the program is a subroutine to a FORTRAN program (for example), the start address is not required because FORTRAN has supplied one.

2.6.10 ENTRY/PUBLIC

ENTRY <name>[,<name>...]
or
PUBLIC <name>[,<name>...]

ENTRY or PUBLIC declares each name in the list as internal and therefore available for use by this program and other programs to be loaded concurrently. All of the names in the list must be defined in the current program or a U error results. An M error is generated if the name is an external name or common-blockname.

2.6.11 EQU

`<name> EQU <exp>`

EQU assigns the value of `<exp>` to `<name>`. If `<exp>` is external, an error is generated. If `<name>` already has a value other than `<exp>`, an M error is generated.

2.6.12 EXT/EXTRN

`EXT <name>[,<name>...]`
or
`EXTRN <name>[,<name>...]`

EXT or EXTRN declares that the name(s) in the list are external (i.e., defined in a different program). If any item in the list references a name that is defined in the current program, an M error results. A reference to a name where the name is followed immediately by two pound signs (e.g., NAME##) also declares the name as external.

2.6.13 INCLUDE

`INCLUDE <filename>`

The INCLUDE pseudo-op applies only to CP/M versions of MACRO-80. The pseudo-ops INCLUDE, \$INCLUDE and MACLIB are synonymous.

The INCLUDE pseudo-op assembles source statements from an alternate source file into the current source file. Use of INCLUDE eliminates the need to repeat an often-used sequence of statements in the current source file.

`<filename>` is any valid specification, as determined by the operating system. Defaults for filename extensions and device names are the same as those in a MACRO-80 command line.

The INCLUDE file is opened and assembled into the current source file immediately following the INCLUDE statement. When end-of-file is reached, assembly resumes with the statement following INCLUDE.

On a MACRO-80 listing, a plus sign is printed between the assembled code and the source line on each line assembled from an INCLUDE file. (See Section 2.12.)

Nested INCLUDEs are not allowed. If encountered, they will result in an objectionable syntax error 'O'.

The file specified in the operand field must exist. If the file is not found, the error 'V' (value error) is given, and the INCLUDE is ignored.

2.6.14 NAME

NAME ('modname')

NAME defines a name for the module. Only the first six characters are significant in a module name. A module name may also be defined with the TITLE pseudo-op. In the absence of both the NAME and TITLE pseudo-ops, the module name is created from the source file name.

2.6.15 ORG - Define Origin

ORG <exp>

The location counter is set to the value of <exp> and the assembler assigns generated code starting with that value. All names used in <exp> must be known on pass 1, and the value must either be absolute or in the same area as the location counter.

2.6.16 PAGE

PAGE [<exp>]

PAGE causes the assembler to start a new output page. The value of <exp>, if included, becomes the new page size (measured in lines per page) and must be in the range 10 to 255. The default page size is 50 lines per page. The assembler puts a form feed character in the listing file at the end of a page.

2.6.17 SET

<name> SET <exp>

SET is the same as EQU, except no error is generated if <name> is already defined.

2.6.18 SUBTTL

SUBTTL <text>

SUBTTL specifies a subtitle to be listed on the line after the title (see TITLE, Section 2.6.19) on each page heading. <text> is truncated after 60 characters. Any number of SUBTTLS may be given in a program.

2.6.19 TITLE

TITLE <text>

TITLE specifies a title to be listed on the first line of each page. If more than one TITLE is given, a Q error results. The first six characters of the title are used as the module name unless a NAME pseudo operation is used. If neither a NAME or TITLE pseudo-op is used, the module name is created from the source filename.

2.6.20 .COMMENT

.COMMENT <delim><text><delim>

The first non-blank character encountered after .COMMENT is the delimiter. The following <text> comprises a comment block which continues until the next occurrence of <delimiter> is encountered. For example, using an asterisk as the delimiter, the format of the comment block would be:

```
.COMMENT *
any amount of text entered
here as the comment block
.
.
.
*
;return to normal mode
```


2.6.21 .PRINTX

```
.PRINTX <delim><text><delim>
```

The first non-blank character encountered after .PRINTX is the delimiter. The following text is listed on the terminal during assembly until another occurrence of the delimiter is encountered. .PRINTX is useful for displaying progress through a long assembly or for displaying the value of conditional assembly switches. For example:

```
IF      CPM
.PRINTX /CPM version/
ENDIF
```

NOTE

.PRINTX will output on both passes. If only one printout is desired, use the IF1 or IF2 pseudo-op. For example:

```
IF2
IF CPM
.PRINTX /CPM version/
ENDIF
ENDIF
```

will only print if CPM is true
and M80 is in pass 2.

2.6.22 .RADIX

```
.RADIX <exp>
```

The default base (or radix) for all constants is decimal. The .RADIX statement allows the default radix to be changed to any base in the range 2 to 16. For example:

```
MOVI    BX,0FFH
.RADIX  16
MOVI    BX,0FF
```

The two MOVIs in the example are identical. The <exp> in a .RADIX statement is always in decimal radix, regardless of the current radix.

2.6.23 .Z80

.Z80 enables the assembler to accept Z80 opcodes. This is the default condition when the assembler is running on a Z80 operating system. Z80 mode may also be set by appending the Z switch to the MACRO-80 command string -- see Section 2.2.2.

2.6.24 .8080

.8080 enables the assembler to accept 8080 opcodes. This is the default condition when the assembler is running on an 8080 operating system. 8080 mode may also be set by appending the I switch to the MACRO-80 command string -- see Section 2.2.2.

2.6.25 .REQUEST

.REQUEST <filename>[,<filename>...]

.REQUEST sends a request to the LINK-80 loader to search the filenames in the list for undefined globals. The filenames in the list should be in the form of legal symbols. They should not include filename extensions or disk specifications. LINK-80 supplies a default extension and assumes the default disk drive.

2.6.26 Conditional Pseudo Operations

The conditional pseudo operations are:

IF/IFT <exp>	True if <exp> is not 0.
IFE/IFF <exp>	True if <exp> is 0.
IF1	True if pass 1.
IF2	True if pass 2.
IFDEF <symbol>	True if <symbol> is defined or has been declared External.
IFNDEF <symbol>	True if <symbol> is undefined or not declared External.
IFB <arg>	True if <arg> is blank. The angle brackets around <arg> are required.
IFNB <arg>	True if <arg> is not blank. Used for testing when dummy parameters are supplied. The angle brackets around <arg> are required.
IFIDN <arg1>,<arg2>	True if the string <arg1> is IDEntical to the string <arg2>. The angle brackets around <arg1> and <arg2> are required.
IFDIF <arg1>,<arg2>	True if the string <arg1> is DIfferent from the string <arg2>. The angle brackets around <arg1> and <arg2> are required.

All conditionals use the following format:

```
IFxx    [argument]
      .
      .
      .
[ELSE
      .
      .
      .
ENDIF    ]
```

Conditionals may be nested to any level. Any argument to a conditional must be known on pass 1 to avoid V errors and incorrect evaluation. For IF, IFT, IFF, and IFE the expression must involve values which were previously defined and the expression must be absolute. If the name is defined after an IFDEF or IFNDEF, pass 1 considers the name to be undefined, but it will be defined on pass 2.

2.6.26.1 ELSE - Each conditional pseudo operation may optionally be used with the ELSE pseudo operation which allows alternate code to be generated when the opposite condition exists. Only one ELSE is permitted for a given IF, and an ELSE is always bound to the most recent, open IF. A conditional with more than one ELSE or an ELSE without a conditional will cause a C error.

2.6.26.2 ENDIF - Each IF must have a matching ENDIF to terminate the conditional. Otherwise, an 'Unterminated conditional' message is generated at the end of each pass. An ENDIF without a matching IF causes a C error.

2.6.27 Listing Control Pseudo Operations

Output to the listing file can be controlled by two pseudo-ops:

.LIST and .XLIST

If a listing is not being made, these pseudo-ops have no effect. .LIST is the default condition. When a .XLIST is encountered, source and object code will not be listed until a .LIST is encountered.

The output of false conditional blocks is controlled by three pseudo-ops: .SFCOND, .LFCOND, and .TFCOND.

These pseudo-ops give the programmer control over four cases.

1. Normally list false conditionals
For this case, the programmer simply allows the default mode to control the listing. The default mode is list false conditionals. If the programmer decides to suppress false conditionals, the /X switch can be issued in the command line instead of editing the source file.

2. Normally suppress false conditionals
For this case, the programmer issues the `.TFCOND` pseudo-op in the program file. `.TFCOND` reverses (toggles) the default, causing false conditionals to be suppressed. If the programmer decides to list false conditionals, the `/X` switch can be issued in the command line instead of editing the source file.
3. Always suppress/list false conditionals
For these cases, the programmer issues either the `.SFCOND` pseudo-op to always suppress false conditionals, or the `.LFCOND` pseudo-op to always list all false conditionals.
4. Suppress/list some false conditionals
For this case, the programmer has decided for most false conditionals whether to list or suppress, but for some false conditionals the programmer has not yet decided. For the false conditionals decided about, use `.SFCOND` or `.LFCOND`. For those not yet decided, use `.TFCOND`. `.TFCOND` sets the current and default settings to the opposite of the default. Initially, the default is set by giving `/X` or no `/X` in the command line. Two subcases exist:
 1. The programmer wants some false conditionals not to list unless `/X` is given. The programmer uses the `.SFCOND` and `.LFCOND` pseudo-ops to control which areas always suppress or list false conditionals. To selectively suppress some false conditionals, the programmer issues `.TFCOND` at the beginning of the conditional block and again at the end of the conditional block. (NOTE: The second `.TFCOND` should be issued so that the default setting will be the same as the initial setting. Leaving the default equal to the initial setting makes it easier to keep track of the default mode if there are many such areas.) If the conditional block evaluates as false, the lines will be suppressed. In this subcase, issuing the `/X` switch in the command line causes the conditional block affected by `.TFCOND` to list even if it evaluates as false.

2. The programmer wants some false conditionals to list unless /X is given. Two consecutive .TFCONDs places the conditional listing setting in initial state which is determined by the presence or absence of the /X switch in the command line (the first .TFCOND sets the default to not initial; the second to initial). The selected conditional block then responds to the /X switch: if a /X switch is issued in the command line, the conditional block is suppressed if false; if no /X switch is issued in the command line, the conditional block is listed even if false.

The programmer then must reissue the .SFCOND or .LFCOND conditional listing pseudo-op to restore the suppress or list mode. Simply issuing another .TFCOND will not restore the prior mode, but will toggle the default setting. Since in this subcase, the next area of code is supposed to list or suppress false conditionals always, the programmer must issue .SFCOND or .LFCOND.

The three conditional listing pseudo-ops are summarized below.

<u>PSEUDO-OP</u>	<u>DEFINITION</u>
.SFCOND	Suppresses the listing of conditional blocks that evaluate as false.
.LFCOND	Restores the listing of conditional blocks that evaluate as false.
.TFCOND	Toggles the current setting which controls the listing false conditionals. .TFCOND sets the current and default setting to not default. If a /X switch is given in the MACRO-80 run command line for a file which contains .TFCOND, /X reverses the effect of .TFCOND.

The following chart illustrates the effects of the three pseudo-ops when encountered under /X and under no /X.

<u>PSEUDO-OP</u>	<u>NO /X</u>	<u>/X</u>
(none)	ON	OFF
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.LFCOND	ON	ON
.	.	.
.	.	.
.TFCOND	OFF	ON
.	.	.
.	.	.
.TFCOND	ON	OFF
.	.	.
.	.	.
.SFCOND	OFF	OFF
.	.	.
.	.	.
.TFCOND	OFF	ON
.TFCOND	ON	OFF
.	.	.
.	.	.
.TFCOND	OFF	ON

The output of cross reference information is controlled by .CREF and .XCREF. If the cross reference facility (see Chapter 3) has not been invoked, .CREF and .XCREF have no effect. The default condition is .CREF. When a .XCREF is encountered, no cross reference information is output until .CREF is encountered.

The output of MACRO/REPT/IRP/IRPC expansions is controlled by three pseudo-ops: .LALL, .SALL, and .XALL. .LALL lists the complete macro text for all expansions. .SALL suppresses listing of all text and object code produced by macros. .XALL is the default condition; a source line is listed only if it generates object code.

2.6.28 Relocation Pseudo Operations

The ability to create relocatable modules is one of the major features of Microsoft assemblers. Relocatable modules offer the advantages of easier coding and faster testing, debugging and modifying. In addition, it is possible to specify segments of assembled code that will later be loaded into RAM (the Data Relative segment) and ROM/PROM (the Code Relative segment). The pseudo operations that select relocatable areas are CSEG and DSEG. The ASEG pseudo-op is used to generate non-relocatable (absolute) code. The COMMON pseudo-op creates a common data area for every COMMON block that is named in the program.

The default mode for the assembler is Code Relative. That is, assembly begins with a CSEG automatically executed and the location counter in the Code Relative mode, pointing to location 0 in the Code Relative segment of memory. All subsequent instructions will be assembled into the Code Relative segment of memory until an ASEG or DSEG or COMMON pseudo-op is executed. For example, the first DSEG encountered sets the location counter to location zero in the Data Relative segment of memory. The following code is assembled in the Data Relative mode, that is, it is assigned to the Data Relative segment of memory. If a subsequent CSEG is encountered, the location counter will return to the next free location in the Code Relative segment and so on.

The ASEG, DSEG, CSEG pseudo-ops never have operands. If you wish to alter the current value of the location counter, use the ORG pseudo-op.

2.6.28.1 ORG Pseudo-op - At any time, the value of the location counter may be changed by use of the the ORG pseudo-op. The form of the ORG statement is:

ORG <exp>

where the value of <exp> will be the new value of the location counter in the current mode. All names used in <exp> must be known on pass 1 and the value of <exp> must be either Absolute or in the current mode of the location counter. For example, the statements

DSEG
ORG 50

set the Data Relative location counter to 50, relative to the start of the Data Relative segment of memory.

2.6.28.2 LINK-80 - The LINK-80 linking loader (see Chapter 4 of this manual) combines the segments and creates each relocatable module in memory when the program is loaded. The origins of the relocatable segments are not fixed until the program is loaded and the origins are assigned by LINK-80. The command to LINK-80 may contain user-specified origins through the use of the /P (for Code Relative) and /D (for Data and COMMON segments) switches.

For example, a program that begins with the statements

```
ASEG
ORG      800H
```

and is assembled entirely in Absolute mode will always load beginning at 800 unless the ORG statement is changed in the source file. However, the same program, assembled in Code Relative mode with no ORG statement, may be loaded at any specified address by appending the /P:<address> switch to the LINK-80 command string.

2.6.29 Relocation Before Loading

Two pseudo-ops, .PHASE and .DEPHASE, allow code to be located in one area, but executed only at a different, specified area.

For example:

```
0000'      .PHASE 100H
0100      E8 0003   FOO:  CALL   BAZ
0103      E9 FF01   JMP     ZOO
0106      C3       BAZ:  RET
          .DEPHASE
0007'      E9 FFFB   ZOO:  JMP     5
```

All labels within a .PHASE block are defined as the absolute value from the origin of the phase area. The code, however, is loaded in the current area (i.e., from 0' in this example). The code within the block can later be moved to 100H and executed.

2.7 MACROS AND BLOCK PSEUDO OPERATIONS

The macro facilities provided by MACRO-80 include three repeat pseudo operations: repeat (REPT), indefinite repeat (IRP), and indefinite repeat character (IRPC). A macro definition operation (MACRO) is also provided. Each of these four macro operations is terminated by the ENDM pseudo operation.

2.7.1 Terms

For the purposes of discussion of macros and block operations, the following terms will be used:

1. <dummy> is used to represent a dummy parameter. All dummy parameters are legal symbols that appear in the body of a macro expansion.
2. <dummylist> is a list of <dummy>s separated by commas.
3. <arglist> is a list of arguments separated by commas. <arglist> must be delimited by angle brackets. Two angle brackets with no intervening characters (<>) or two commas with no intervening characters enter a null argument in the list. Otherwise an argument is a character or series of characters terminated by a comma or >. With angle brackets that are nested inside an <arglist>, one level of brackets is removed each time the bracketed argument is used in an <arglist>. See example, Section 2.7.5.) A quoted string is an acceptable argument and is passed as such. Unless enclosed in brackets or a quoted string, leading and trailing spaces are deleted from arguments.
4. <paramlist> is used to represent a list of actual parameters separated by commas. No delimiters are required (the list is terminated by the end of line or a comment), but the rules for entering null parameters and nesting brackets are the same as described for <arglist>. (See example, Section 2.7.5)

2.7.2 REPT-ENDM

```
REPT <exp>
.
.
.
ENDM
```

The block of statements between REPT and ENDM is repeated <exp> times. <exp> is evaluated as a 16-bit unsigned number. If <exp> contains any external or undefined terms, an error is generated. Example:

```
SET 0
REPT 10      ;generates DB 1 - DB 10
SET X+1
DB X
ENDM
```

2.7.3 IRP-ENDM

```
IRP <dummy>,<arglist>
.
.
.
ENDM
```

The <arglist> must be enclosed in angle brackets. The number of arguments in the <arglist> determines the number of times the block of statements is repeated. Each repetition substitutes the next item in the <arglist> for every occurrence of <dummy> in the block. If the <arglist> is null (i.e., <>), the block is processed once with each occurrence of <dummy> removed. For example:

```
IRP X,<1,2,3,4,5,6,7,8,9,10>
DB X
ENDM
```

generates the same bytes as the REPT example.

2.7.4 IRPC-ENDM

```
IRPC <dummy>,string (or <string>)
.
.
.
ENDM
```

IRPC is similar to IRP but the arglist is replaced by a string of text and the angle brackets around the string are optional. The statements in the block are repeated once for each character in the string. Each repetition substitutes the next character in the string for every occurrence of <dummy> in the block. For example:

```
IRPC X,0123456789
DB X+1
ENDM
```

generates the same code as the two previous examples.

2.7.5 MACRO

Often it is convenient to be able to generate a given sequence of statements from various places in a program, even though different parameters may be required each time the sequence is used. This capability is provided by the MACRO statement.

The form is

```
<name> MACRO <dummylist>
      .
      .
      .
      ENDM
```

where <name> conforms to the rules for forming symbols. <name> is the name that will be used to invoke the macro. The <dummy>s in <dummylist> are the parameters that will be changed (replaced) each time the MACRO is invoked. The statements before the ENDM comprise the body of the macro. During assembly, the macro is expanded every time it is invoked but, unlike REPT/IRP/IRPC, the macro is not expanded when it is encountered.

The form of a macro call is

```
<name> <paramlist>
```

where <name> is the name supplied in the MACRO definition, and the parameters in <paramlist> will replace the <dummy>s in the MACRO <dummylist> on a one-to-one basis. The number of items in <dummylist> and <paramlist> is limited only by the length of a line. The number of parameters used when the macro is called need not be the same as the number of <dummy>s in <dummylist>. If there are more parameters than <dummy>s, the extras are ignored. If there are fewer, the extra <dummy>s will be made null. The assembled code will contain the macro expansion code after each macro call.

NOTE

A dummy parameter in a MACRO/REPT/IRP/IRPC is always recognized exclusively as a dummy parameter. Register names such as A and B will be changed in the expansion if they were used as dummy parameters.

Here is an example of a MACRO definition that defines a macro called FOO:

```
FOO      MACRO      X
Y        SET        0
          REPT       X
Y        SET        Y+1
          DB         Y
          ENDM
          ENDM
```

This macro generates the same code as the previous three examples when the call

```
FOO      10
```

is executed.

Another example, which generates the same code, illustrates the removal of one level of brackets when an argument is used as an arglist:

```
FOO      MACRO      X
          IRP        Y,<X>
          DB         Y
          ENDM
          ENDM
```

When the call

```
FOO      <1,2,3,4,5,6,7,8,9,10>
```

is made, the macro expansion looks like this:

```
IRP      Y,<1,2,3,4,5,6,7,8,9,10>
DB       Y
ENDM
```

2.7.6 ENDM

Every REPT, IRP, IRPC and MACRO pseudo-op must be terminated with the ENDM pseudo-op. Otherwise, the 'Unterminated REPT/IRP/IRPC/MACRO' message is generated at the end of each pass. An unmatched ENDM causes an O error.

2.7.7 EXITM

The EXITM pseudo-op is used to terminate a REPT/IRP/IRPC or MACRO call. When an EXITM is executed, the expansion is exited immediately and any remaining expansion or repetition is not generated. If the block containing the EXITM is nested within another block, the outer level continues to be expanded.

2.7.8 LOCAL

LOCAL <dummylist>

The LOCAL pseudo-op is allowed only inside a MACRO definition. When LOCAL is executed, the assembler creates a unique symbol for each <dummy> in <dummylist> and substitutes that symbol for each occurrence of the <dummy> in the expansion. These unique symbols are usually used to define a label within a macro, thus eliminating multiply-defined labels on successive expansions of the macro. The symbols created by the assembler range from ..0001 to ..FFFF. Users will therefore want to avoid the form ..nnnn for their own symbols. If LOCAL statements are used, they must be the first statements in the macro definition.

2.7.9 Special Macro Operators And Forms

- & The ampersand is used in a macro expansion to concatenate text or symbols. A dummy parameter that is in a quoted string will not be substituted in the expansion unless it is immediately preceded by &. To form a symbol from text and a dummy, put & between them. For example:

```
ERRGEN MACRO X
ERROR&X:PUSH BX
      MOVI BX,'&X'
      JMP ERROR
      ENDM
```

In this example, the call ERRGEN A will generate:

```
ERRORA: PUSH B
      MOVI BX,'A'
      JMP ERROR
```

- ;; In a block operation, a comment preceded by two semicolons is not saved as part of the expansion (i.e., it will not appear on the listing even under .LALL). A comment preceded by one semicolon, however, will be preserved and appear in the expansion.
- ! When an exclamation point is used in an argument, the next character is entered literally (i.e., !; and <;> are equivalent).

NUL NUL is an operator that returns true if its argument (a parameter) is null. The remainder of a line after NUL is considered to be the argument to NUL. The conditional

IF NUL argument

is false if, during the expansion, the first character of the argument is anything other than a semicolon or carriage return. It is recommended that testing for null parameters be done using the IFB and IFNB conditionals.

% The percent sign is used only in a macro argument. % converts the expression that follows it (usually a symbol) to a number in the current radix. During macro expansion, the number derived from converting the expression is substituted for the dummy. Using the % special operator allows a macro call by value. (Usually, a macro call is a call by reference with the text of the macro argument substituting exactly for the dummy.)

The expression following the % must conform to the same rules as the DS (Define Space) pseudo-op. A valid expression returning a non-relocatable constant is required.

EXAMPLE: Normally, LB, the argument to MAKLAB, would be substituted for Y, the argument to MACRO, as a string. The % causes LB to be converted to a non-relocatable constant which is then substituted for Y. Without the % special operator, the result of assembly would be 'Error LB' rather than 'Error 1', etc.

MAKLAB	MACRO	Y
ERR&Y:	DB	'Error &Y',0
	ENDM	
MAKERR	MACRO	X
LB	SET	0
	REPT	X
LB	SET	LB+1
	MAKLAB	%LB
	ENDM	
	ENDM	

When called by MAKERR 3, the assembler will generate:

ERR1:	DB	'Error 1',0
ERR2:	DB	'Error 2',0
ERR3:	DB	'Error 3',0

TYPE The TYPE operator returns a byte that describes two characteristics of its argument: 1) the mode, and 2) whether it is External or not. The argument to TYPE may be any expression (string, numeric, logical). If the expression is invalid, TYPE returns zero.

The byte that is returned is configured as follows:

The lower two bits are the mode. If the lower two bits are:

0	the mode is Absolute
1	the mode is Program Relative
2	the mode is Data Relative
3	the mode is Common Relative

The high bit (80H) is the External bit. If the high bit is on, the expression contains an External. If the high bit is off, the expression is local (not External).

The Defined bit is 20H. This bit is on if the expression is locally defined, and it is off if the expression is undefined or external. If neither bit is on, the expression is invalid.

TYPE is usually used inside macros, where an argument type may need to be tested to make a decision regarding program flow. For example:

FOO	MACRO	X
	LOCAL	Z
Z	SET TYPE	X
IF	Z...	

2.8 USING Z80 PSEUDO-OPS

When using the MACRO-80 assembler, the following Z80 pseudo-ops are valid. The function of each pseudo-op is equivalent to that of its counterpart.

<u>Z80 pseudo-op</u>	<u>Equivalent pseudo-op</u>
COND	IFT
ENDC	ENDIF
*EJECT	PAGE
DEFB	DB
DEFS	DS
DEFW	DW
DEFM	DB
DEFL	SET
GLOBAL	PUBLIC
EXTERNAL	EXTRN

The formats, where different, conform to the previous format. That is, DEFB and DEFW are permitted a list of arguments (as are DB and DW), and DEFM is permitted a string or numeric argument (as is DB).

2.9 SAMPLE ASSEMBLY

A>M80

*EXMPL1,TTY:=EXMPL1

MAC80 3.2

PAGE 1

```

00100 ;CSL3(P1,P2)
00200 ;SHIFT P1 LEFT CIRCULARLY 3 BITS
00300 ;RETURN RESULT IN P2
00400 ENTRY CSL3
00450 ;GET VALUE OF FIRST PARAMETER
00500 CSL3:
0000' 7E 00600 MOV A,M
0001' 23 00700 INX H
0002' 66 00800 MOV H,M
0003' 6F 00900 MOV L,A
01000 ;SHIFT COUNT
0004' 06 03 01100 MVI B,3
0006' AF 01200 LOOP: XRA A
01300 ;SHIFT LEFT
0007' 29 01400 DAD H
01500 ;ROTATE IN CY BIT
0008' 17 01600 RAL
0009' 85 01700 ADD L
000A' 6F 01800 MOV L,A
01900 ;DECREMENT COUNT
000B' 05 02000 DCR B
02100 ;ONE MORE TIME
000C' C2 0006' 02200 JNZ LOOP
000F' EB 02300 XCHG
02400 ;SAVE RESULT IN SECOND PARAMETER
0010' 73 02500 MOV M,E
0011' 23 02600 INX H
0012' 72 02700 MOV M,D
0013' C9 02800 RET
02900 END

```

MAC80 3.2

PAGE S

CSL3 0000I' LOOP 0006'

No Fatal error(s)

2.10 MACRO-80 ERRORS

MACRO-80 errors are indicated by a one-character flag in column one of the listing file. If a listing file is not being printed on the terminal, each erroneous line is also printed or displayed on the terminal. Below is a list of the MACRO-80 Error Codes:

- A Argument error
Argument to pseudo-op is not in correct format or is out of range (.PAGE 1; .RADIX 1; PUBLIC 1; JMPS TOOFAR).
- C Conditional nesting error
ELSE without IF, ENDIF without IF, two ELSEs on one IF.
- D Double Defined symbol
Reference to a symbol which is multiply defined.
- E External error
Use of an external illegal in context (e.g., FOO SET NAME##; MOVI AX,2-NAME##).
- M Multiply Defined symbol
Definition of a symbol which is multiply defined.
- N Number error
Error in a number, usually a bad digit (e.g., 8Q).
- O Bad opcode or objectionable syntax
ENDM, LOCAL outside a block; SET, EQU or MACRO without a name; bad syntax in an opcode; or bad syntax in an expression (mismatched parenthesis, quotes, consecutive operators, etc.).
- P Phase error
Value of a label or EQU name is different on pass 2.
- Q Questionable
Usually means a line is not terminated properly. This is a warning error (e.g. MOV AX,BX,).
- R Relocation
Illegal use of relocation in expression, such as abs-rel. Data, code and COMMON areas are relocatable.
- U Undefined symbol
A symbol referenced in an expression is not defined. (For certain pseudo-ops, a V error is printed on pass 1 and a U on pass 2.)

V Value error
On pass 1 a pseudo-op which must have its value known on pass 1 (e.g., .RADIX, .PAGE, DS, IF, IFE, etc.), has a value which is undefined. If the symbol is defined later in the program, a U error will not appear on the pass 2 listing.

Error Messages:

'No end statement encountered on input file'
No END statement: either it is missing or it is not parsed due to being in a false conditional, unterminated IRP/IRPC/REPT block or terminated macro.

'Unterminated conditional'
At least one conditional is unterminated at the end of the file.

'Unterminated REPT/IRP/IRPC/MACRO'
At least one block is unterminated.

[xx] [No] Fatal error(s) [,xx warnings]
The number of fatal errors and warnings. The message is listed on the CRT and in the list file.

2.11 COMPATIBILITY WITH OTHER ASSEMBLERS

The \$EJECT and \$TITLE controls are provided for compatibility with INTEL's ISIS assembler. The dollar sign must appear in column 1 only if spaces or tabs separate the dollar sign from the control word. The control

\$EJECT

is the same as the MACRO-80 PAGE pseudo-op.
The control

\$TITLE('text')

is the same as the MACRO-80 SUBTTL <text> pseudo-op.

The INTEL operands PAGE and INPAGE generate Q errors when used with the MACRO-80 CSEG or DSEG pseudo-ops. These errors are warnings; the assembler ignores the operands.

When MACRO-80 is entered, the default for the origin is Code Relative 0.

With the INTEL ISIS assembler, the default is Absolute 0.

With MACRO-80, the dollar sign (\$) is a defined constant that indicates the value of the location counter at the start of the statement. Other assemblers may use a decimal point or an asterisk. Other constants are defined by MACRO-80 to have the following values:

A=7	B=0	C=1	D=2	E=3
H=4	L=5	M=6	SP=6	PSW=6

2.12 FORMAT OF LISTINGS

On each page of a MACRO-80 listing, the first two lines have the form:

```
[TITLE text]      M80 3.3      PAGE x[-y]
[SUBTTL text]
```

where:

1. TITLE text is the text supplied with the TITLE pseudo-op, if one was given in the source program.
2. x is the major page number, which is incremented only when a form feed is encountered in the source file. (When using Microsoft's EDIT-80 text editor, a form feed is inserted whenever a page mark is done.) When the symbol table is being printed, x = S.
3. y is the minor page number, which is incremented whenever the .PAGE pseudo-op is encountered in the source file, or whenever the current page size has been filled.
4. SUBTTL text is the text supplied with the SUBTTL pseudo-op, if one was given in the source program.

Next, a blank line is printed, followed by the first line of output.

A line of output on a MACRO-80 listing has the following form:

```
[crf#]      [error] loc#m   |xx | xxxx|...   source
```

If cross reference information is being output, the first item on the line is the cross reference number, followed by a tab.

A one-letter error code followed by a space appears next on the line, if the line contains an error. If there is no error, a space is printed. If there is no cross reference number, the error code column is the first column on the listing.

The value of the location counter appears next on the line. It is a 4-digit hexadecimal number or 6-digit octal number, depending on whether the /O or /H switch was given in the MACRO-80 command string.

The character at the end of the location counter value is the mode indicator. It will be one of the following symbols:

'	Code Relative
"	Data Relative
!	COMMON Relative
<space>	Absolute
*	External

Next, three spaces are printed followed by the assembled code. One-byte values are followed by a space. Two-byte values are followed by a mode indicator. Two-byte values are printed in the opposite order they are stored in, i.e., the high order byte is printed first. Externals are either the offset or the value of the pointer to the next External in the chain.

If a line of output on a MACRO-80 listing is from an INCLUDE file, the character 'C' is printed after the assembled code on that line. If a line of output is part of a text expansion (MACRO, REPT, IRP, IRPC) a plus sign '+' is printed after the assembled code on that line.

The remainder of the line contains the line of source code, as it was input.

Example:

```
0C49 3A A912' C+ LDA LCOUNT
```

'C+' indicates this line is from an INCLUDE file and part of a macro expansion.

2.12.1 Symbol Table Listing

In the symbol table listing, all the macro names in the program are listed alphabetically, followed by all the symbols in the program, listed alphabetically. After each symbol, a tab is printed, followed by the value of the symbol. If the symbol is Public, an I is printed immediately after the value. The next character printed will be one of the following:

U	Undefined symbol.
C	COMMON block name. (The "value" of the COMMON block is its length (number of bytes) in hexadecimal or octal.)
*	External symbol.
<space>	Absolute value.
'	Program Relative value.
"	Data Relative value.
!	COMMON Relative value.

CHAPTER 4

LINK-80 LINKING LOADER

NOTE

If you are using the TEKDOS operating system, see Appendix A for proper command formats.

4.1 RUNNING LINK-80

The command to run LINK-80 is

L80

LINK-80 returns the prompt "*", indicating it is ready to accept commands.

4.2 COMMAND FORMAT

Each command to LINK-80 consists of a string of object filenames separated by commas. These are the files to be loaded by LINK-80. The command format is:

objfile1,objfile2,...objfilen

The default extension for all filenames is REL. Command lines are supported, that is, the invocation and command may be typed on the same line.

Example:

L80 MYPROG,YRPROG

Any filename in the LINK-80 command string can also specify a device name. The default device name with the CP/M operating system is the currently logged disk. The default device with the ISIS-II operating system is disk drive 0. The format is:

dev1:objfile1,dev2:objfile2,...devn:objfilen

The device names are as listed in Section 2.2.1.

Example:

L80 MYPROG,A:YRPROG

After each line is typed, LINK-80 will load the specified files. After LINK finishes this process, it will list all symbols that remained undefined followed by an asterisk.

Example:

*MAIN

DATA 0100 0200

SUBR1* (SUBR1 is undefined)

*SUBR1

DATA 0100 0300

*

Typically, to execute a MACRO-80 program and subroutines, the user types the list of filenames followed by /G (begin execution). To resolve any external, undefined symbols, you can first search your library routines (See Chapter 5, LIB-80) by appending the filenames, followed by /S, to the loader command string.

*MYLIB/S Searches MYLIB.REL for unresolved global symbols

*/G Starts execution

4.2.1 LINK-80 Switches

A number of switches may be given in the LINK-80 command string to specify actions affecting the loading or execution of the program(s). Each switch must be preceded by a slash (/). (With the TEKDOS operating system, switches are preceded by hyphens. See Appendix A.)

Switches may be placed wherever applicable in the command string:

1. At command level. It is possible for a switch to be the entire LINK-80 command, or to appear first in the command string. For example:

`*/G` Tells LINK-80 to begin execution of program(s) already loaded

`*/M` List all global references from program(s) already loaded

`*/P:200,FOO` Load FOO, with program area beginning at address 200

2. Immediately after a filename. An S or N switch may refer to only one filename in the command string. Therefore, when the S or N switch is required, it is placed immediately after that filename, regardless of where the filename appears in the command string. For example:

`*MYLIB/S,MYPROG`
Search MYLIB.REL and load necessary library modules, then load MYPROG.REL.

`*MYPROG/N,MYPROG/E`
Load MYPROG.REL, save MYPROG.COM on disk and exit LINK-80.

3. At the end of the command string. Any required switches that affect the entire load process may be appended at the end of the command string. For example:

`*MYPROG/N,MYPROG/M/E`
Open a CP/M COM file called MYPROG.COM, load MYPROG.REL and list all global references. Exit LINK-80 and save the COM file.

`MYLIB/S,MYSUB,MYPROG/N,MYPROG/M/G`
Search MYLIB.REL, load and link MYSUB.REL and MYPROG.REL, open a CP/M COM file called MYPROG.COM, list all global references, save the COM file, and execute MYPROG.

The available switches are:

<u>Switch</u>	<u>Action</u>
R	Reset. Put loader back in its initial state. Use /R if you loaded the wrong file by mistake and want to restart. /R takes effect as soon as it is encountered in a command string.
E or E:Name	Exit LINK-80 and return to the operating system. The system library will be searched on the current disk to satisfy any existing undefined globals. Before exiting, LINK-80 prints three numbers: the start address, the address of the next available byte, and the number of 256-byte pages used. The optional form E:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program. Use /E to load a program and exit back to the monitor.
G or G:Name	Start execution of the program as soon as the current command line has been interpreted. The system library will be searched on the current disk to satisfy any existing undefined globals if they exist. Before execution actually begins, LINK-80 prints three numbers and a BEGIN EXECUTION message. The three numbers are the start address, the address of the next available byte, and the number of 256-byte pages used. The optional form G:Name (where Name is a global symbol previously defined in one of the modules) uses Name for the start address of the program.
N	If a <filename>/N is specified, the program will be saved on disk under the selected name (with a default extension of .COM for CP/M) when a /E or /G is done. A jump to the start of the program is inserted if needed so the program can run properly (at 100H for CP/M).

P and D

/P and /D allow the origin(s) to be set for the next program loaded. /P and /D take effect when seen (not deferred), and they have no effect on programs already loaded. The form is /P:<address> or /D:<address>, where <address> is the desired origin in the current typeout radix. (Default radix is hex. /O sets radix to octal; /H to hex.) LINK-80 does a default /P:<link origin>+3 (i.e., 103H for CP/M and 4003H for ISIS) to leave room for the jump to the start address. NOTE: Do not use /P or /D to load programs or data into the locations of the loader's jump to the start address (100H to 102H for CP/M) unless it is to load the start of the program there. If programs or data are loaded into these locations, the jump will not be generated.

If no /D is given, data areas are loaded before program areas for each module. If a /D is given, all Data and Common areas are loaded starting at the data origin and the program area at the program origin. Example:

```
*/P:200,FOO
Data      200      300
*/R
*/P:200 /D:400,FOO
Data      400      480
Program 200      280
```

U

List the origin and end of the program and data area and all undefined globals as soon as the current command line has been interpreted. The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

M

List the origin and end of the program and data area, all defined globals and their values, and all undefined globals followed by an asterisk. The program information is only printed if a /D has been done. Otherwise, the program is stored in the data area.

S

Search the filename immediately preceding the /S in the command string to satisfy any undefined globals.

4.2.2 CP/M LINK-80 Switches

The following switches apply to CP/M versions only.

X If a filename/N was specified, /X will cause the file to be saved in Intel ASCII HEX format with an extension of HEX.

Example: FOO/N/X/E will create an Intel ASCII HEX formatted load module named FOO.HEX.

Y If a filename/N was specified, /Y will create a filename.SYM file when /E is entered. This file contains the names and addresses of all Globals for use with Digital Research's Symbolic Debugger, SID and ZSID.

Example: FOO/N/Y/E creates FOO.COM and FOO.SYM. MYPROG/N/X/Y/E creates MYPROG.HEX and MYPROG.SYM.

4.2.3 Sample LinksLINK AND GO

```
A>L80
*EXAMPL,EXMPL1/G
DATA      3000      30AC
[304F      30AC      49]
[BEGIN EXECUTION]

          1792          14336
          14336          -16383
        -16383           14
           14           112
           112           896

A>
```

LINK AND SAVE

```
A>L80
*EXAMPL,EXAMPL1,EXAM/N/E
DATA      3000      30AC
[304F      30AC      49]
A>
```

Loads and links EXAMPL.REL, EXMPL1.REL and creates EXAM.COM.

4.3 FORMAT OF LINK COMPATIBLE OBJECT FILES

NOTE

Section 4.3 is reference material for users who wish to know the load format of LINK-80 relocatable object files. Most users will want to skip this section, as it does not contain material necessary to the operation of the package.

LINK-compatible object files consist of a bit stream. Individual fields within the bit stream are not aligned on byte boundaries, except as noted below. Use of a bit stream for relocatable object files keeps the size of object files to a minimum, thereby decreasing the number of disk reads/writes.

There are two basic types of load items: Absolute and Relocatable. The first bit of an item indicates one of these two types. If the first bit is a 0, the following 8 bits are loaded as an absolute byte. If the first bit is a 1, the next 2 bits are used to indicate one of four types of relocatable items:

- 00 Special LINK item (see below).
- 01 Program Relative. Load the following 16 bits after adding the current Program base.
- 10 Data Relative. Load the following 16 bits after adding the current Data base.
- 11 Common Relative. Load the following 16 bits after adding the current Common base.

Special LINK items consist of the bit stream 100 followed by:

a four-bit control field

an optional A field consisting of a two-bit address type that is the same as the two-bit field above except 00 specifies absolute address

an optional B field consisting of 3 bits that give a symbol length and up to 8 bits for each character of the symbol

A general representation of a special LINK item is:

1	00	xxxx	yy	nn	zzz + characters of symbol name
			<u>A field</u>		<u>B field</u>

xxxx	Four-bit control field (0-15 below)
yy	Two-bit address type field
nn	Sixteen-bit value
zzz	Three-bit symbol length field

The following special types have a B-field only:

0	Entry symbol (name for search)
1	Select COMMON block
2	Program name
3	Request library search
4	Extension LINK items (see below)

The following special LINK items have both an A field and a B field:

5	Define COMMON size
6	Chain external (A is head of address chain, B is name of external symbol)
7	Define entry point (A is address, B is name)

The following special LINK items have an A field only:

8	External - offset. Used for JMP and CALL to externals
9	External + offset. The A value will be added to the two bytes starting at the current location counter immediately before execution.
10	Define size of Data area (A is size)
11	Set loading location counter to A
12	Chain address. A is head of chain, replace all entries in chain with current location counter. The last entry in the chain has an address field of absolute zero.
13	Define program size (A is size)
14	End program (forces to byte boundary)

The following special Link item has neither an A nor a B field:

15 End file

An Extension LINK item follows the general format of a B-field-only special LINK item, but contents of the B-field are not a symbol name. Instead, the symbol area contains one character to identify the type of Extension LINK item, followed by from 1 to 7 characters of additional information.

Thus, every Extension LINK item has the format:

1 00 0100 zzz i jjjjjjj

where

zzz may be any three bit integer (with 000 representing 8),

i is an eight bit Extension LINK item type identifier, and

jjjjjjj are zzz-1 eight bit characters of information whose significance depends on i

At present, there is only one Extension LINK item:

i = X'35' COBOL overlay segment sentinel

zzz = 010 (binary)

j = COBOL segment number -49 (decimal)

When the overlay segment sentinel is encountered by the linker, the current overlay segment number is set to the value of j+49. If the previously existing segment number was non-zero and a /N switch is in effect, the data area is written to disk in a file whose name is the current program name and whose extension is Vnn, where nn are the two hexadecimal digits representing the number j+49 (decimal).

4.4 LINK-80 ERROR MESSAGES

LINK-80 has the following error messages:

?No Start Address A /G switch was issued, but no main program had been loaded.

?Loading Error The last file given for input was not a properly formatted LINK-80 object file.

?Out of Memory Not enough memory to load program.

?Command Error Unrecognizable LINK-80 command.

?<file> Not Found <file>, as given in the command string, did not exist.

%2nd COMMON Larger /XXXXXX/
The first definition of COMMON block /XXXXXX/ was not the largest definition. Reorder module loading sequence or change COMMON block definitions.

%Mult. Def. Global YYYYYY
More than one definition for the global (internal) symbol YYYYYY was encountered during the loading process.

%Overlaying { Program } Area ,Start = xxxx
 { Data } ,Public = <symbol name>(xxxx)
 ,External = <symbol name>(xxxx)
A /D or /P will cause already loaded data to be destroyed.

?Intersecting { Program } Area
 { Data }
The program and data area intersect and an address or external chain entry is in this intersection. The final value cannot be converted to a current value since it is in the area intersection.

?Start Symbol - <name> - Undefined
After a /E: or /G: is given, the symbol specified was not defined.

Origin { Above } Loader Memory, Move Anyway (Y or N)?
 { Below }

After a /E or /G was given, either the data or program area has an origin or top which lies outside loader memory (i.e., loader origin to top of memory). If a Y <cr> is given, LINK-80 will move the area and continue. If anything else is given, LINK-80 will exit. In either case, if a /N was given, the image will already have been saved.

?Can't Save Object File

A disk error occurred when the file was being saved.

4.5 PROGRAM BREAK INFORMATION

LINK-80 stores the address of the first free location in a global symbol called \$MEMORY if that symbol has been defined by a program loaded. \$MEMORY is set to the top of the data area +1.

NOTE

If /D is given and the data origin is less than the program area, the user must be sure there is enough room to keep the program from being destroyed. This is particularly true with the disk driver for FORTRAN-80 which uses \$MEMORY to allocate disk buffers and FCB's.

1942

...

...

...

...

...

APPENDIX A

TEKDOS Operating System

The command formats for MACRO-80, LINK-80 and CREF-80 differ slightly under the TEKDOS operating system.

A.1 TEKDOS COMMAND FILES

The files F80, M80, and C80 are actually TEKDOS command files for the compiler, assembler, loader, and cross reference programs, respectively. These command files set the emulation mode to 0 and select the Z-80 assembler processor (see TEKDOS documentation), then execute the appropriate program file. You will note that all of these command files are set up to execute the Microsoft programs from drive 1. LINK-80 will also look for the library (FORLIB) on drive 1. If you wish to execute any of this software from drive 0, the command file must be edited and LINK-80 should be given an explicit library search directive "FORLIB-S". (See Section 4.2.1 of this manual.)

A.2 MACRO-80

The M80 assembler accepts command lines only. A prompt is not displayed and interactive commands are not accepted. Commands have the same format as TEKDOS assembler commands; i.e., three filename or device name parameters plus optional switches.

M80 [objfile] [lstfile] sourcefile [sw1] [sw2...]

The object and listing file parameters are optional. These files will not be created if the parameters are omitted, however any error messages will still be displayed on the console. The available switches are as described in Chapter 2 of this manual. except that the switches are delimited by commas or spaces instead of slashes.

A.3 CREF-80

The form of commands to CREF80 is:

```
C80 1stfile sourcefile
```

Both filename parameters are required. The sourcefile parameter is always the name of a CREF80 file created during assembly, by use of the C switch.

Example:

Create a CREF80 file using MACRO-80:

```
M80 ,, TSTCRF TSTMAC C
```

Create a cross reference listing from the CREF80 file:

```
C80 TSTLST TSTCRF
```

A.4 LINK-80

With TEKDOS, the LINK-80 loader accepts interactive commands only. Command lines are not supported.

When LINK-80 is invoked, and whenever it is waiting for input, it will prompt with an asterisk. Commands are lists of filenames and/or devices separated by commas or spaces and optionally interspersed with switches. The input to LINK-80 must be Microsoft relocatable object code (not the same as TEKDOS loader format).

Switches to LINK-80 are delimited by hyphens under TEKDOS, instead of slashes. All LINK-80 switches (as documented in Chapter 3) are supported, except "G" and "N", which are not implemented at this time.

Examples:

1. Assemble a MACRO-80 program named XTEST, creating an object file called XREL and a listing file called XLST:

```
>M80 XREL XLST XTEST
```

2. Load XTEST and save the loaded module:

```
>L80
*XREL-E
[04AD 22B8]
*DOS*ERROR 46
L80 TERMINATED
>M XMOD 400 22B8 04AD
```

Note that "-E" exits via an error message due to execution of a Halt instruction. The memory image is intact, however, and the "Module" command may be used to save it. Once a program is saved in module format, it may then be executed directly without going through LINK-80 again.

The bracketed numbers printed by LINK-80 before exiting are the entry point address and the highest address loaded, respectively. The loader default is to begin loading at 400H. However, the loader also places a jump to the start address in location 0, thereby allowing execution to begin at 0. The memory locations between 0003 and 0400H are reserved for SRB's and I/O buffers at runtime.

INDEX

\$INCLUDE	2-14
\$MEMRY	4-11
.COMMENT	2-16
.CREF	2-23
.DEPHASE	2-25
.LALL	2-23
.LFCOND	2-20
.LIST	2-20
.PAGE	2-37
.PHASE	2-25
.PRINTX	2-17
.RADIX	2-6, 2-17
.REQUEST	2-18
.SALL	2-23
.SFCOND	2-20
.TFCOND	2-20
.XALL	2-23
.XCREF	2-23
.XLIST	2-20
Absolute memory	2-8, 2-11, 2-38
Arithmetic operators	2-8
ASEG	2-8, 2-11, 2-24
Block pseudo ops	2-25
Character constants	2-7
Code Relative	2-11, 2-24 to 2-25, 2-38
Command format	2-1, 3-1, 4-1, 5-1
Comments	2-6, 2-16
COMMON	2-8, 2-11, 2-24 to 2-25, 2-38 to 2-39
Conditionals	2-19
Constants	2-6
CP/M	2-2 to 2-3, 4-4 to 4-6, 5-1, 5-4
Cross reference facility	2-4, 2-23, 2-37, 3-1
CSEG	2-8, 2-11, 2-24, 2-36
Data Relative	2-8, 2-12, 2-24 to 2-25, 2-38
DB	2-6, 2-11
DC	2-12
Define Byte	2-6, 2-11
Define Character	2-12
Define Origin	2-15
Define Space	2-12
Define Word	2-13
DS	2-12
DSEG	2-8, 2-12, 2-24, 2-36
DW	2-13

EDIT-80	2-5, 2-37
ELSE	2-20
END	2-13
ENDIF	2-20
ENDM	2-25, 2-29
ENTRY	2-13, 5-2
EQU	2-14 to 2-15
Error codes	2-35, 2-37
Error messages	2-36, 4-10
EXITM	2-29
EXT	2-14
Externals	2-9, 2-14, 2-35, 2-38
EXTRN	2-14
IF	2-19
IF1	2-19
IF2	2-19
IFB	2-19
IFDEF	2-19
IFDIF	2-19
IFE	2-19
IFF	2-19
IFIDN	2-19
IFNB	2-19
IFT	2-19
INCLUDE	2-14
INTEL	2-36
IRP	2-23, 2-25, 2-27
IRPC	2-23, 2-25, 2-27
ISIS-II	2-2 to 2-3, 2-5, 4-5
LIB-80	5-1
Library manager	5-1
LINK-80	2-11, 2-13, 2-18, 2-25, 4-1, 5-4
Listings	2-14, 2-20, 2-37 to 2-38, 3-2, 5-4
LOCAL	2-30
Logical operators	2-8
MACLIB	2-14
MACRO	2-23, 2-25 to 2-26, 2-28 to 2-29
Macro operators	2-30
Modes	2-8
Modules	5-2
NAME	2-15
Operators	2-8
ORG	2-11, 2-13, 2-15, 2-24
PAGE	2-15, 2-36
Program Relative	2-8
PUBLIC	2-5, 2-13, 2-39
REPT	2-23, 2-25 to 2-26
SET	2-15

Strings 2-7
SUBTTL 2-16, 2-36 to 2-37
Switches 2-3, 3-1, 4-2, 5-3, 5-5
Symbol table 2-37, 2-39

TEKDOS 2-1, 3-1, 4-1, A-1
TITLE 2-15 to 2-16, 2-37

Microsoft Software Problem Report

Use this form to report errors or problems in: ☐ FORTRAN-80

☐ COBOL-80

☐ MACRO-80

☐ LINK-80

Release (or version) number: _____

Date _____

Report only one problem per form.

Describe your hardware and operating system: _____

Please supply a concise description of the problem and the circumstances surrounding its occurrence. If possible, reduce the problem to a simple test case. Otherwise, include all programs and data in machine readable form (preferably on a diskette). If a patch or interim solution is being used, please describe it.

This form may also be used to describe suggested enhancements to Microsoft software.

Problem Description:

Did you find errors in the documentation supplied with the software? If so, please include page numbers and describe:

Fill in the following information before returning this form:

Name _____ Phone _____

Organization _____

Address _____ City _____ State _____ Zip _____

Return form to:

Microsoft
10800 NE Eighth, Suite 819
Bellevue, WA 98004



Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

ED: A CONTEXT EDITOR FOR THE CP/M DISK SYSTEM
USER'S MANUAL

COPYRIGHT (c) 1976, 1978

DIGITAL RESEARCH

Copyright (c) 1976, 1978 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

Table of Contents

1.	ED TUTORIAL	1
1.1	Introduction to ED	1
1.2	ED Operation	1
1.3	Text Transfer Functions	1
1.4	Memory Buffer Organization	5
1.5	Memory Buffer Operation	5
1.6	Command Strings	7
1.7	Text Search and Alteration	8
1.8	Source Libraries	11
1.9	Repetitive Command Execution	12
2.	ED ERROR CONDITIONS	13
3.	CONTROL CHARACTERS AND COMMANDS	14

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

1/12/1964

ED USER'S MANUAL

1. ED TUTORIAL

1.1. Introduction to ED.

ED is the context editor for CP/M, and is used to create and alter CP/M source files. ED is initiated in CP/M by typing

$$\text{ED} \left\{ \begin{array}{l} \langle \text{filename} \rangle \\ \langle \text{filename} \rangle . \langle \text{filetype} \rangle \end{array} \right\}$$

In general, ED reads segments of the source file given by `<filename>` or `<filename> . <filetype>` into central memory, where the file is manipulated by the operator, and subsequently written back to disk after alterations. If the source file does not exist before editing, it is created by ED and initialized to empty. The overall operation of ED is shown in Figure 1.

1.2. ED Operation

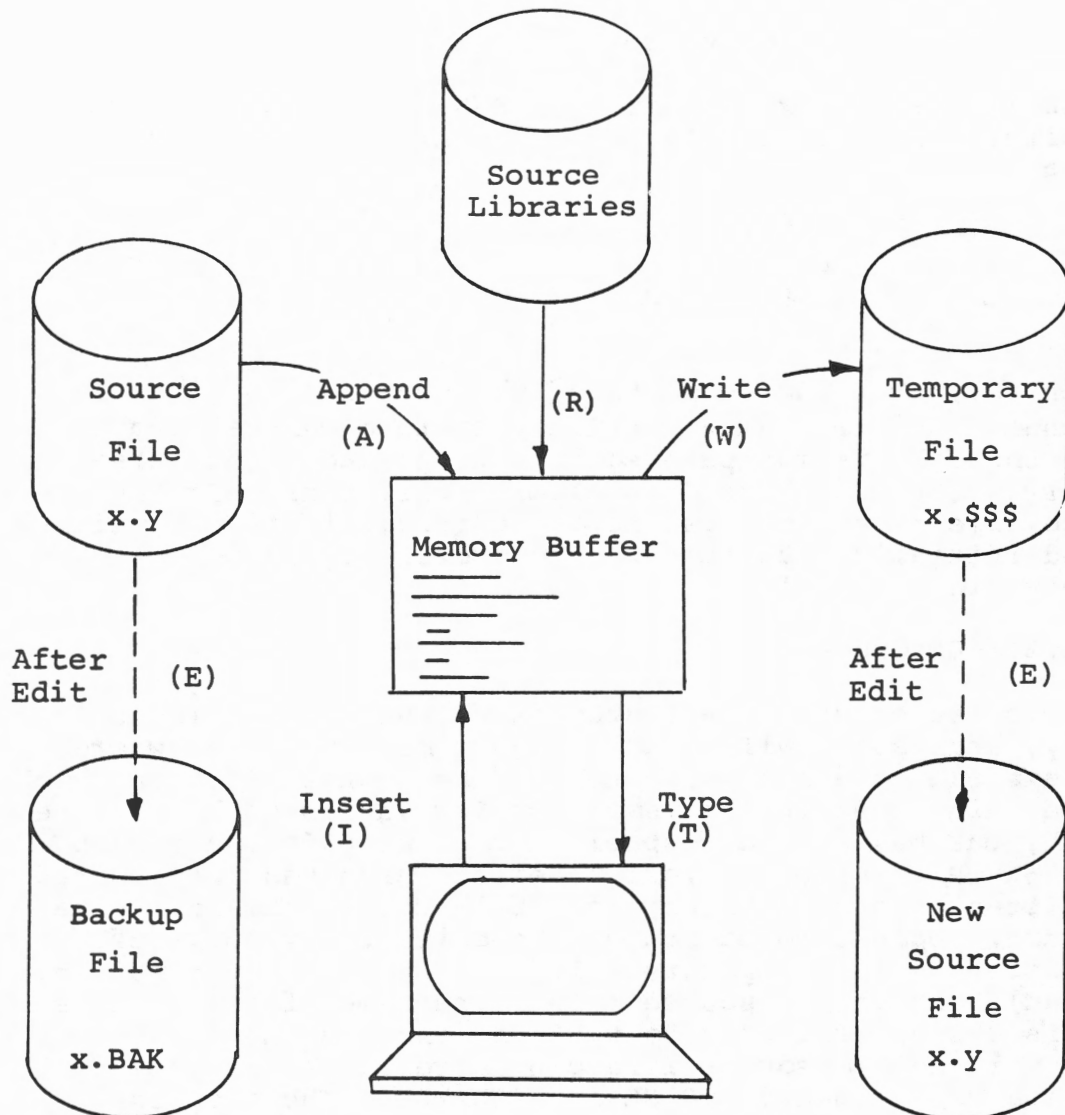
ED operates upon the source file, denoted in Figure 1 by `x.y`, and passes all text through a memory buffer where the text can be viewed or altered (the number of lines which can be maintained in the memory buffer varies with the line length, but has a total capacity of about 6000 characters in a 16K CP/M system). Text material which has been edited is written onto a temporary work file under command of the operator. Upon termination of the edit, the memory buffer is written to the temporary file, followed by any remaining (unread) text in the source file. The name of the original file is changed from `x.y` to `x.BAK` so that the most recent previously edited source file can be reclaimed if necessary (see the CP/M commands `ERASE` and `RENAME`). The temporary file is then changed from `x.$$$` to `x.y` which becomes the resulting edited file.

The memory buffer is logically between the source file and working file as shown in Figure 2.

1.3. Text Transfer Functions

Given that `n` is an integer value in the range 0 through 65535, the following ED commands transfer lines of text from the source file through the memory buffer to the temporary (and eventually final) file:

Figure 1. Overall ED Operation



Note: the ED program accepts both lower and upper case ASCII characters as input from the console. Single letter commands can be typed in either case. The U command can be issued to cause ED to translate lower case alphabets to upper case as characters are filled to the memory buffer from the console. Characters are echoed as typed without translation, however. The -U command causes ED to revert to "no translation" mode. ED starts with an assumed -U in effect.

Figure 2. Memory Buffer Organization

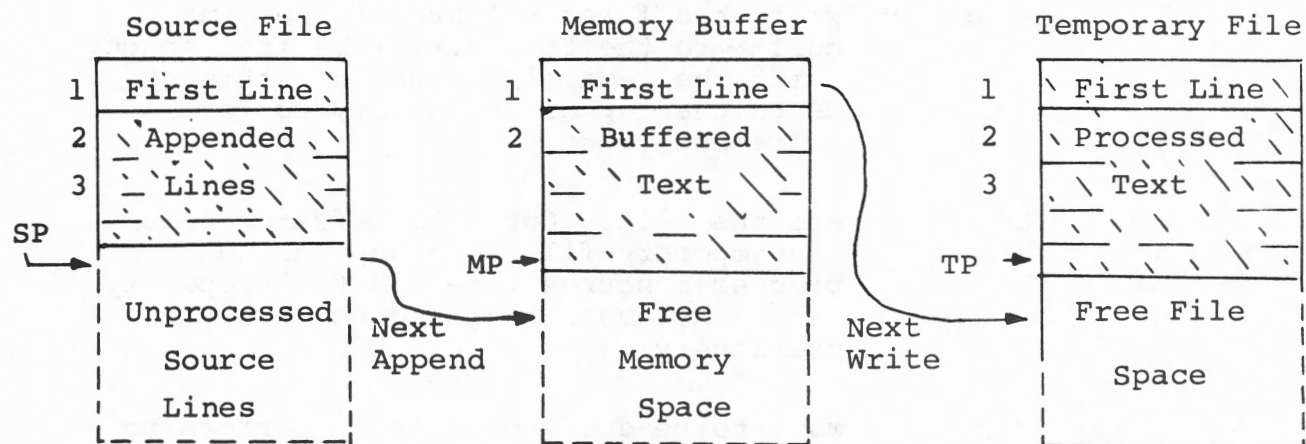
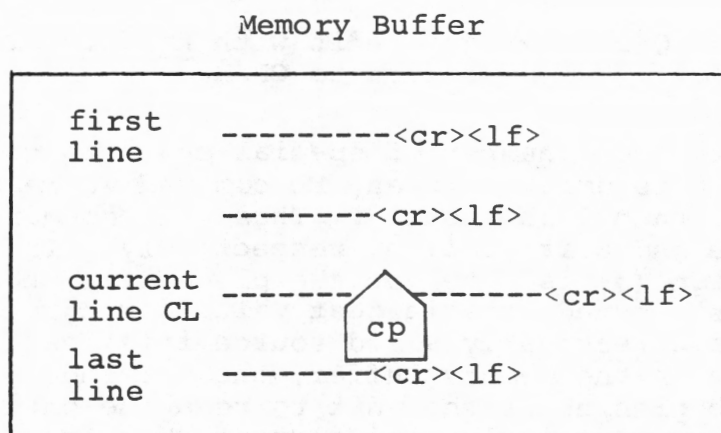


Figure 3. Logical Organization of Memory Buffer



- nA<cr>* - append the next n unprocessed source lines from the source file at SP to the end of the memory buffer at MP. Increment SP and MP by n.

- nW<cr> - write the first n lines of the memory buffer to the temporary file free space. Shift the remaining lines n+1 through MP to the top of the memory buffer. Increment TP by n.

- E<cr> - end the edit. Copy all buffered text to temporary file, and copy all unprocessed source lines to the temporary file. Rename files as described previously.

- H<cr> - move to head of new file by performing automatic E command. Temporary file becomes the new source file, the memory buffer is emptied, and a new temporary file is created (equivalent to issuing an E command, followed by a reinvocation of ED using x.y as the file to edit).

- O<cr> - return to original file. The memory buffer is emptied, the temporary file is deleted, and the SP is returned to position 1 of the source file. The effects of the previous editing commands are thus nullified.


- Q<cr> - quit edit with no file alterations, return to CP/M.

There are a number of special cases to consider. If the integer n is omitted in any ED command where an integer is allowed, then 1 is assumed. Thus, the commands A and W append one line and write 1 line, respectively. In addition, if a pound sign (#) is given in the place of n, then the integer 65535 is assumed (the largest value for n which is allowed). Since most reasonably sized source files can be contained entirely in the memory buffer, the command #A is often issued at the beginning of the edit to read the entire source file to memory. Similarly, the command #W writes the entire buffer to the temporary file. Two special forms of the A and W

*<cr> represents the carriage-return key

commands are provided as a convenience. The command 0A fills the current memory buffer to at least half-full, while 0W writes lines until the buffer is at least half empty. It should also be noted that an error is issued if the memory buffer size is exceeded. The operator may then enter any command (such as W) which does not increase memory requirements. The remainder of any partial line read during the overflow will be brought into memory on the next successful append.

1.4. Memory Buffer Organization

The memory buffer can be considered a sequence of source lines brought in with the A command from a source file. The memory buffer has an associated (imaginary) character pointer CP which moves throughout the memory buffer under command of the operator. The memory buffer appears logically as shown in Figure 3 where the dashes represent characters of the source line of indefinite length, terminated by carriage-return (<cr>) and line-feed (<lf>) characters, and  represents the imaginary character pointer. Note that the CP is always located ahead of the first character of the first line, behind the last character of the last line, or between two characters. The current line CL is the source line which contains the CP.

1.5. Memory Buffer Operation

Upon initiation of ED, the memory buffer is empty (ie, CP is both ahead and behind the first and last character). The operator may either append lines (A command) from the source file, or enter the lines directly from the console with the insert command

I<cr>

ED then accepts any number of input lines, where each line terminates with a <cr> (the <lf> is supplied automatically), until a control-z (denoted by ↑z is typed by the operator. The CP is positioned after the last character entered. The sequence

```
I<cr>
NOW IS THE<cr>
TIME FOR<cr>
ALL GOOD MEN<cr>
↑z
```

leaves the memory buffer as shown below

```
NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf>
```



Various commands can then be issued which manipulate the CP or display source text in the vicinity of the CP. The commands shown below with a preceding n indicate that an optional unsigned value can be specified. When preceded by +, the command can be unsigned, or have an optional preceding plus or minus sign. As before, the pound sign (#) is replaced by 65535. If an integer n is optional, but not supplied, then n=1 is assumed. Finally, if a plus sign is optional, but none is specified, then + is assumed.

- +B<cr> - move CP to beginning of memory buffer if +, and to bottom if -.
- +nC<cr> - move CP by +n characters (toward front of buffer if +), counting the <cr><lf> as two distinct characters
- +nD<cr> - delete n characters ahead of CP if plus and behind CP if minus.
- +nK<cr> - kill (ie remove) +n lines of source text using CP as the current reference. If CP is not at the beginning of the current line when K is issued, then the characters before CP remain if + is specified, while the characters after CP remain if - is given in the command.
- +nL<cr> - if n=0 then move CP to the beginning of the current line (if it is not already there) if n≠0 then first move the CP to the beginning of the current line, and then move it to the beginning of the line which is n lines down (if +) or up (if -). The CP will stop at the top or bottom of the memory buffer if too large a value of n is specified.

`±nT<cr>` - If `n=0` then type the contents of the current line up to CP. If `n=1` then type the contents of the current line from CP to the end of the line. If `n>1` then type the current line along with `n-1` lines which follow, if `+` is specified. Similarly, if `n>1` and `-` is given, type the previous `n` lines, up to the CP. The break key can be depressed to abort long type-outs.

`±n<cr>` - equivalent to `±nLT`, which moves up or down and types a single line

1.6. Command Strings

Any number of commands can be typed contiguously (up to the capacity of the CP/M console buffer), and are executed only after the `<cr>` is typed. Thus, the operator may use the CP/M console command functions to manipulate the input command:

Rubout	remove the last character
Control-U	delete the entire line
Control-C	re-initialize the CP/M System
Control-E	return carriage for long lines without transmitting buffer (max 128 chars)

Suppose the memory buffer contains the characters shown in the previous section, with the CP following the last character of the buffer. The command strings shown below produce the results shown to the right

<u>Command String</u>	<u>Effect</u>	<u>Resulting Memory Buffer</u>
1. <code>B2T<cr></code>	move to beginning of buffer and type 2 lines: "NOW IS THE TIME FOR"	<div> <div>CP</div> NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf> </div>
2. <code>5C0T<cr></code>	move CP 5 characters and type the beginning of the line "NOW I"	<div> NOW I <div>CP</div> S THE<cr><lf> </div>

- | | | | |
|----|--|---|---|
| 3. | 2L-T<cr> | move two lines down
and type previous
line
"TIME FOR" | NOW IS THE<cr><lf>
TIME FOR<cr><lf>
ALL GOOD MEN<cr><lf> |
| | | <div style="border: 1px solid black; padding: 2px; display: inline-block;">cp</div> | |
| 4. | -L#K<cr> | move up one line,
delete 65535 lines
which follow | NOW IS THE<cr><lf> |
| | | | <div style="border: 1px solid black; padding: 2px; display: inline-block;">cp</div> |
| 5. | I<cr>
TIME TO<cr>
INSERT<cr>
↑z | insert two lines
of text | NOW IS THE<cr><lf>
TIME TO<cr><lf>
INSERT<cr><lf> |
| | | | <div style="border: 1px solid black; padding: 2px; display: inline-block;">cp</div> |
| 6. | -2L#T<cr> | move up two lines,
and type 65535
lines ahead of CP
"NOW IS THE" | NOW IS THE<cr><lf>
TIME TO<cr><lf>
INSERT<cr><lf> |
| | | | <div style="border: 1px solid black; padding: 2px; display: inline-block;">cp</div> |
| 7. | <cr> | move down one line
and type one line
"INSERT" | NOW IS THE<cr><lf>
TIME TO<cr><lf>
INSERT<cr><lf> |
| | | | <div style="border: 1px solid black; padding: 2px; display: inline-block;">cp</div> |

1.7. Text Search and Alteration




ED also has a command which locates strings within the memory buffer. The command takes the form

$$nF \ c_1 c_2 \dots c_k \left\{ \begin{array}{c} \text{<cr>} \\ \uparrow z \end{array} \right\}$$

where c_1 through c_k represent the characters to match followed by either a <cr> or control -z*. ED starts at the current position of CP and attempts to match all k characters. The match is attempted n times, and if successful, the CP is moved directly after the character c_k . If the n matches are not successful, the CP is not moved from its initial position. Search strings can include ↑l (control-l), which is replaced by the pair of symbols <cr><lf>.

*The control-z is used if additional commands will be typed following the ↑z.

The following commands illustrate the use of the F command:


<u>Command String</u>	<u>Effect</u>	<u>Resulting Memory Buffer</u>
1. B#T<cr>	move to beginning and type entire buffer	 NOW IS THE<cr><lf> TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>
2. FS T<cr>	find the end of the string "S T"	NOW IS T  HE<cr><lf>
3. FI↑z0TT	find the next "I" and type to the CP then type the remainder of the current line: "TIME FOR"	NOW IS THE<cr><lf> TI  ME FOR<cr><lf> ALL GOOD MEN<cr><lf>

An abbreviated form of the insert command is also allowed, which is often used in conjunction with the F command to make simple textual changes. The form is:

$I c_1 c_2 \dots c_n \uparrow z$ or

$I c_1 c_2 \dots c_n <cr>$

where c_1 through c_n are characters to insert. If the insertion string is terminated by a $\uparrow z$, the characters c_1 through c_n are inserted directly following the CP, and the CP is moved directly after character c_n . The action is the same if the command is followed by a $<cr>$ except that a $<cr><lf>$ is automatically inserted into the text following character c_n . Consider the following command sequences as examples of the F and I commands:

<u>Command String</u>	<u>Effect</u>	<u>Resulting Memory Buffer</u>
BITHIS IS $\uparrow z <cr>$	Insert "THIS IS " at the beginning of the text	THIS IS NOW THE $<cr><lf>$  TIME FOR<cr><lf> ALL GOOD MEN<cr><lf>

FTIME↑z-4DIPLACE↑z<cr>

find "TIME" and delete
it; then insert "PLACE"

THIS IS NOW THE<cr><lf>

PLACE  FOR<cr><lf>
ALL GOOD MEN<cr><lf>

3FO↑z-3D5DICHANGES↑<cr>


find third occurrence
of "O" (ie the second
"O" in GOOD), delete
previous 3 characters;
then insert "CHANGES"

THIS IS NOW THE <cr><lf>

PLACE FOR<cr><lf>
ALL CHANGES  <cr><lf>

-8CISOURCE<cr> move back 8 characters
and insert the line
"SOURCE<cr><lf>"

THIS IS NOW THE<cr><lf>

PLACE FOR<cr><lf>
ALL SOURCE<cr><lf>
 CHANGES<cr><lf>

ED also provides a single command which combines the F and I commands to perform simple string substitutions. The command takes the form

$$n \ S \ c_1 c_2 \dots c_k \uparrow z \ d_1 d_2 \dots d_m \left\{ \begin{array}{c} <cr> \\ \uparrow z \end{array} \right\}$$

and has exactly the same effect as applying the command string

$$F \ c_1 c_2 \dots c_k \uparrow z -kDI d_1 d_2 \dots d_m \left\{ \begin{array}{c} <cr> \\ \uparrow z \end{array} \right\}$$

a total of n times. That is, ED searches the memory buffer starting at the current position of CP and successively substitutes the second string for the first string until the end of buffer, or until the substitution has been performed n times.

As a convenience, a command similar to F is provided by ED which automatically appends and writes lines as the search proceeds. The form is

$$n \ N \ c_1 c_2 \dots c_k \left\{ \begin{array}{c} cr \\ \uparrow z \end{array} \right\}$$

which searches the entire source file for the nth occurrence of the string $c_1 c_2 \dots c_k$ (recall that F fails if the string cannot be found in the current buffer). The operation of the

N command is precisely the same as F except in the case that the string cannot be found within the current memory buffer. In this case, the entire memory contents is written (ie, an automatic #W is issued). Input lines are then read until the buffer is at least half full, or the entire source file is exhausted. The search continues in this manner until the string has been found n times, or until the source file has been completely transferred to the temporary file.

A final line editing function, called the juxtaposition command takes the form

$$n \ J \ c_1 c_2 \dots c_k \uparrow z \ d_1 d_2 \dots d_m \uparrow z \ e_1 e_2 \dots e_q \left\{ \begin{array}{l} \langle cr \rangle \\ \uparrow z \end{array} \right\}$$

with the following action applied n times to the memory buffer: search from the current CP for the next occurrence of the string $c_1 c_2 \dots c_k$. If found, insert the string $d_1 d_2 \dots d_m$, and move CP to follow d_m . Then delete all characters following CP up to (but not including) the string $e_1 e_2 \dots e_q$, leaving CP directly after d_m . If $e_1 e_2 \dots e_q$ cannot be found, then no deletion is made. If the current line is

cp NOW IS THE TIME<cr><lf>

Then the command

JW $\uparrow z$ WHAT $\uparrow z \uparrow l$ <cr>

Results in

NOW WHAT cp <cr><lf>

(Recall that $\uparrow l$ represents the pair <cr><lf> in search and substitute strings).

It should be noted that the number of characters allowed by ED in the F, S, N, and J commands is limited to 100 symbols.

1.8. Source Libraries

ED also allows the inclusion of source libraries during the editing process with the R command. The form of this command is

$R f_1 f_2 \dots f_n \uparrow z$ or

$R f_1 f_2 \dots f_n \langle cr \rangle$

where $f_1 f_2 \dots f_n$ is the name of a source file on the disk with as assumed filetype of 'LIB'. ED reads the specified file, and places the characters into the memory buffer after CP, in a manner similar to the I command. Thus, if the command

RMACRO $\langle cr \rangle$

is issued by the operator, ED reads from the file MACRO.LIB until the end-of-file, and automatically inserts the characters into the memory buffer.

1.9. Repetitive Command Execution

The macro command M allows the ED user to group ED commands together for repeated evaluation. The M command takes the form:

$$n \ M \ c_1 c_2 \dots c_k \ \left\{ \begin{array}{c} \langle cr \rangle \\ \uparrow z \end{array} \right\}$$

where $c_1 c_2 \dots c_k$ represent a string of ED commands, not including another M command. ED executes the command string n times if $n > 1$. If $n = 0$ or 1, the command string is executed repetitively until an error condition is encountered (e.g., the end of the memory buffer is reached with an F command).

As an example, the following macro changes all occurrences of GAMMA to DELTA within the current buffer, and types each line which is changed:

MFGAMMA $\uparrow z$ -5DIDELTA $\uparrow z$ 0TT $\langle cr \rangle$

or equivalently

MSGAMMA $\uparrow z$ DELTA $\uparrow z$ 0TT $\langle cr \rangle$

2. ED ERROR CONDITIONS

On error conditions, ED prints the last character read before the error, along with an error indicator:

?	unrecognized command
>	memory buffer full (use one of the commands D,K,N,S, or W to remove characters), F,N, or S strings too long.
#	cannot apply command the number of times specified (e.g., in F command)
O	cannot open LIB file in R command

Cyclic redundancy check (CRC) information is written with each output record under CP/M in order to detect errors on subsequent read operations. If a CRC error is detected, CP/M will type

PERM ERR DISK d

where d is the currently selected drive (A,B,...). The operator can choose to ignore the error by typing any character at the console (in this case, the memory buffer data should be examined to see if it was incorrectly read), or the user can reset the system and reclaim the backup file, if it exists. The file can be reclaimed by first typing the contents of the BAK file to ensure that it contains the proper information:

TYPE x.BAK<cr>

where x is the file being edited. Then remove the primary file:

ERA x.y<cr>

and rename the BAK file:

REN x.y=x.BAK<cr>

The file can then be re-edited, starting with the previous version.

3. CONTROL CHARACTERS AND COMMANDS

The following table summarizes the control characters and commands available in ED:

<u>Control Character</u>	<u>Function</u>
↑c	system reboot
↑e	physical <cr><lf> (not actually entered in command)
↑i	logical tab (cols 1,8,15,...)
↑l	logical <cr><lf> in search and substitute strings
↑u	line delete
↑z	string terminator
rubout	character delete
break	discontinue command (e.g., stop typing)

<u>Command</u>	<u>Function</u>
nA	append lines
±B	begin bottom of buffer
±nC	move character positions
±nD	delete characters
E	end edit and close files (normal end)
nF	find string
H	end edit, close and reopen files
I	insert characters
nJ	place strings in juxtaposition
±nK	kill lines
±nL	move down/up lines
nM	macro definition
nN	find next occurrence with autoscan
O	return to original file
±nP	move and print pages
Q	quit with no file changes
R	read library file
nS	substitute strings
±nT	type lines
± U	translate lower to upper case if U, no translation if -U
nW	write lines
nZ	sleep
±n<cr>	move and type (±nLT)

Appendix A: ED 1.4 Enhancements

The ED context editor contains a number of commands which enhance its usefulness in text editing. The improvements are found in the addition of line numbers, free space interrogation, and improved error reporting.

The context editor issued with CP/M 1.4 produces absolute line number prefixes when the "V" (Verify Line Numbers) command is issued. Following the V command, the line number is displayed ahead of each line in the format:

nnnnn:

where nnnnn is an absolute line number in the range 1 to 65535. If the memory buffer is empty, or if the current line is at the end of the memory buffer, then nnnnn appears as 5 blanks.

The user may reference an absolute line number by preceding any command by a number followed by a colon, in the same format as the line number display. In this case, the ED program moves the current line reference to the absolute line number, if the line exists in the current memory buffer. Thus, the command

345:T

is interpreted as "move to absolute line 345, and type the line." Note that absolute line numbers are produced only during the editing process, and are not recorded with the file. In particular, the line numbers will change following a deleted or expanded section of text.

The user may also reference an absolute line number as a backward or forward distance from the current line by preceding the absolute line number by a colon. Thus, the command

:400T

is interpreted as "type from the current line number through the line whose absolute number is 400." Combining the two line reference forms, the command

345::400T

for example, is interpreted as "move to absolute line 345, then type through absolute line 400." Note that absolute line references of this sort can precede any of the standard ED commands.

A special case of the V command, "0V", prints the memory buffer statistics in the form:

free/total

where "free" is the number of free bytes in the memory buffer (in decimal), and "total" is the size of the memory buffer.

ED 1.4 also includes a "block move" facility implemented through the "X" (Xfer) command. The form

nX

transfers the next n lines from the current line to a temporary file called

X\$\$\$\$\$\$\$.LIB

which is active only during the editing process. In general, the user can reposition the current line reference to any portion of the source file and transfer lines to the temporary file. The transferred line accumulate one after another in this file, and can be retrieved by simply typing:

R

which is the trivial case of the library read command. In this case, the entire transferred set of lines is read into the memory buffer. Note that the X command does not remove the transferred lines from the memory buffer, although a K command can be used directly after the X, and the R command does not empty the transferred line file. That is, given that a set of lines has been transferred with the X command, they can be re-read any number of times back into the source file. The command

ØX

is provided, however, to empty the transferred line file.

Note that upon normal completion of the ED program through Q or E, the temporary LIB file is removed. If ED is aborted through ctl-C, the LIB file will exist if lines have been transferred, but will generally be empty (a subsequent ED invocation will erase the temporary file).

Due to common typographical errors, ED 1.4 requires several potentially disastrous commands to be typed as single letters, rather than in composite commands. The commands

E (end), H (head), O (original), Q (quit)

must be typed as single letter commands.

ED 1.4 also prints error messages in the form

BREAK "x" AT c

where x is the error character, and c is the command where the error occurred.

1. The first part of the report deals with the general situation of the country.

2. The second part of the report deals with the economic situation of the country.

3. The third part of the report deals with the social situation of the country.

4. The fourth part of the report deals with the political situation of the country.

5. The fifth part of the report deals with the cultural situation of the country.

6. The sixth part of the report deals with the environmental situation of the country.

7. The seventh part of the report deals with the international situation of the country.

8. The eighth part of the report deals with the future prospects of the country.

9. The ninth part of the report deals with the conclusion of the report.

10. The tenth part of the report deals with the appendix of the report.



DIGITAL RESEARCH®

Post Office Box 579, Pacific Grove, California 93950, (408) 649-3896

CP/M DYNAMIC DEBUGGING TOOL (DDT)

USER'S GUIDE

COPYRIGHT (c) 1976, 1978

DIGITAL RESEARCH

Copyright (c) 1976, 1978 by Digital Research. All rights reserved. No part of this publication may be reproduced, transmitted, transcribed, stored in a retrieval system, or translated into any language or computer language, in any form or by any means, electronic, mechanical, magnetic, optical, chemical, manual or otherwise, without the prior written permission of Digital Research, Post Office Box 579, Pacific Grove, California 93950.

Disclaimer

Digital Research makes no representations or warranties with respect to the contents hereof and specifically disclaims any implied warranties of merchantability or fitness for any particular purpose. Further, Digital Research reserves the right to revise this publication and to make changes from time to time in the content hereof without obligation of Digital Research to notify any person of such revision or changes.

Table of Contents

Section	Page
I. INTRODUCTION	1
II. DDT COMMANDS	3
1. The A (Assemble) Command	3
2. The D (Display) Command	4
3. The F (Fill) Command	4
4. The G (Go) Command	4
5. The I (Input) Command	5
6. The L (List) Command	6
7. The M (Move) Command	6
8. The R (Read) Command	6
9. The S (Set) Command	7
10. The T (Trace) Command	7
11. The U (Untrace) Command	8
12. The X (Examine) Command	8
III. IMPLEMENTATION NOTES	9
IV. AN EXAMPLE	10

CP/M Dynamic Debugging Tool (DDT)

User's Guide

I. Introduction.

The DDT program allows dynamic interactive testing and debugging of programs generated in the CP/M environment. The debugger is initiated by typing one of the following commands at the CP/M Console Command level

```
DDT
DDT filename.HEX
DDT filename.COM
```

where "filename" is the name of the program to be loaded and tested. In both cases, the DDT program is brought into main memory in the place of the Console Command Processor (refer to the CP/M Interface Guide for standard memory organization), and thus resides directly below the Basic Disk Operating System portion of CP/M. The BDOS starting address, which is located in the address field of the JMP instruction at location 5H, is altered to reflect the reduced Transient Program Area size.

The second and third forms of the DDT command shown above perform the same actions as the first, except there is a subsequent automatic load of the specified HEX or COM file. The action is identical to the sequence of commands

```
DDT
Ifilename.HEX or Ifilename.COM
R
```

where the I and R commands set up and read the specified program to test (see the explanation of the I and R commands below for exact details).

Upon initiation, DDT prints a sign-on message in the format

```
nnK DDT-s VER m.m
```

where nn is the memory size (which must match the CP/M system being used), s is the hardware system which is assumed, corresponding to the codes

D	-	Digital Research standard version
M	-	MDS version
I	-	IMSAI standard version
O	-	Omron systems
S	-	Digital Systems standard version

and m.m is the revision number.

Following the sign on message, DDT prompts the operator with the character "-" and waits for input commands from the console. The operator can type any of several single character commands, terminated by a carriage return to execute the command. Each line of input can be line-edited using the standard CP/M controls

rubout	remove the last character typed
ctl-U	remove the entire line, ready for re-typing
ctl-C	system reboot

Any command can be up to 32 characters in length (an automatic carriage return is inserted as the 33rd character), where the first character determines the command type

A	enter assembly language mnemonics with operands
D	display memory in hexadecimal and ASCII
F	fill memory with constant data
G	begin execution with optional breakpoints
I	set up a standard input file control block
L	list memory using assembler mnemonics
M	move a memory segment from source to destination
R	read program for subsequent testing
S	substitute memory values
T	trace program execution
U	untraced program monitoring
X	examine and optionally alter the CPU state

The command character, in some cases, is followed by zero, one, two, or three hexadecimal values which are separated by commas or single blank characters. All DDT numeric output is in hexadecimal form. In all cases, the commands are not executed until the carriage return is typed at the end of the command.

At any point in the debug run, the operator can stop execution of DDT using either a ctl-C or G0 (jmp to location 0000H), and save the current memory image using a SAVE command of the form

SAVE n filename.COM

where n is the number of pages (256 byte blocks) to be saved on disk. The number of blocks can be determined by taking the high order byte of the top load address and converting this number to decimal. For example, if the highest address in the Transient Program Area is 1234H then the number of pages is 12H, or 18 in decimal. Thus the operator could type a ctl-C during the debug run, returning to the Console Processor level, followed by

SAVE 18 X.COM

The memory image is saved as X.COM on the diskette, and can be directly executed by simply typing the name X. If further testing is required, the memory image can be recalled by typing

DDT X.COM

which reloads previously saved program from location 100H through page 18 (12FFH). The machine state is not a part of the COM file, and thus the program must be restarted from the beginning in order to properly test it.

II. DDT COMMANDS.

The individual commands are given below in some detail. In each case, the operator must wait for the prompt character (-) before entering the command. If control is passed to a program under test, and the program has not reached a breakpoint, control can be returned to DDT by executing a RST 7 from the front panel (note that the rubout key should be used instead if the program is executing a T or U command). In the explanation of each command, the command letter is shown in some cases with numbers separated by commas, where the numbers are represented by lower case letters. These numbers are always assumed to be in a hexadecimal radix, and from one to four digits in length (longer numbers will be automatically truncated on the right).

Many of the commands operate upon a "CPU state" which corresponds to the program under test. The CPU state holds the registers of the program being debugged, and initially contains zeroes for all registers and flags except for the program counter (P) and stack pointer (S), which default to 100H. The program counter is subsequently set to the starting address given in the last record of a HEX file if a file of this form is loaded (see the I and R commands).

1. The A (Assemble) Command. DDT allows inline assembly language to be inserted into the current memory image using the A command which takes the form

As

where s is the hexadecimal starting address for the inline assembly. DDT prompts the console with the address of the next instruction to fill, and reads the console, looking for assembly language mnemonics (see the Intel 8080 Assembly Language Reference Card for a list of mnemonics), followed by register references and operands in absolute hexadecimal form. Each successive load address is printed before reading the console. The A command terminates when the first empty line is input from the console.

Upon completion of assembly language input, the operator can review the memory segment using the DDT disassembler (see the L command).

Note that the assembler/disassembler portion of DDT can be overlaid by the transient program being tested, in which case the DDT program responds with an error condition when the A and L commands are used (refer to Section IV).

2. The D (Display) Command. The D command allows the operator to view the contents of memory in hexadecimal and ASCII formats. The forms are

D
Ds
Ds,f

In the first case, memory is displayed from the current display address (initially 100H), and continues for 16 display lines. Each display line takes the form shown below

aaaa bb bb bb bb bb bb bb bb bb bb bb bb bb bb cccccccccccccccc

where aaaa is the display address in hexadecimal, and bb represents data present in memory starting at aaaa. The ASCII characters starting at aaaa are given to the right (represented by the sequence of c's), where non-graphic characters are printed as a period (.) symbol. Note that both upper and lower case alphabets are displayed, and thus will appear as upper case symbols on a console device that supports only upper case. Each display line gives the values of 16 bytes of data, except that the first line displayed is truncated so that the next line begins at an address which is a multiple of 16.

The second form of the D command shown above is similar to the first, except that the display address is first set to address s. The third form causes the display to continue from address s through address f. In all cases, the display address is set to the first address not displayed in this command, so that a continuing display can be accomplished by issuing successive D commands with no explicit addresses.

Excessively long displays can be aborted by pushing the rubout key.

3. The F (Fill) Command. The F command takes the form

Fs,f,c

where s is the starting address, f is the final address, and c is a hexadecimal byte constant. The effect is as follows: DDT stores the constant c at address s, increments the value of s and tests against f. If s exceeds f then the operation terminates, otherwise the operation is repeated. Thus, the fill command can be used to set a memory block to a specific constant value.

4. The G (Go) Command. Program execution is started using the G command, with up to two optional breakpoint addresses. The G command takes one of the forms

G
Gs
Gs,b

Gs,b,c
G,b
G,b,c

The first form starts execution of the program under test at the current value of the program counter in the current machine state, with no breakpoints set (the only way to regain control in DDT is through a RST 7 execution). The current program counter can be viewed by typing an X or XP command. The second form is similar to the first except that the program counter in the current machine state is set to address s before execution begins. The third form is the same as the second, except that program execution stops when address b is encountered (b must be in the area of the program under test). The instruction at location b is not executed when the breakpoint is encountered. The fourth form is identical to the third, except that two breakpoints are specified, one at b and the other at c. Encountering either breakpoint causes execution to stop, and both breakpoints are subsequently cleared. The last two forms take the program counter from the current machine state, and set one and two breakpoints, respectively.

Execution continues from the starting address in real-time to the next breakpoint. That is, there is no intervention between the starting address and the break address by DDT. Thus, if the program under test does not reach a breakpoint, control cannot return to DDT without executing a RST 7 instruction. Upon encountering a breakpoint, DDT stops execution and types

*d

where d is the stop address. The machine state can be examined at this point using the X (Examine) command. The operator must specify breakpoints which differ from the program counter address at the beginning of the G command. Thus, if the current program counter is 1234H, then the commands

G,1234

and

G400,400

both produce an immediate breakpoint, without executing any instructions whatsoever.

5. The I (Input) Command. The I command allows the operator to insert a file name into the default file control block at 5CH (the file control block created by CP/M for transient programs is placed at this location; see the CP/M Interface Guide). The default FCB can be used by the program under test as if it had been passed by the CP/M Console Processor. Note that this file name is also used by DDT for reading additional HEX and COM files. The form of the I command is

Ifilename

or

Ifilename.filetype

If the second form is used, and the filetype is either HEX or COM, then subsequent R commands can be used to read the pure binary or hex format machine code (see the R command for further details).

6. The L (List) Command. The L command is used to list assembly language mnemonics in a particular program region. The forms are

L
Ls
Ls,f

The first command lists twelve lines of disassembled machine code from the current list address. The second form sets the list address to s, and then lists twelve lines of code. The last form lists disassembled code from s through address f. In all three cases, the list address is set to the next unlisted location in preparation for a subsequent L command. Upon encountering an execution breakpoint, the list address is set to the current value of the program counter (see the G and T commands). Again, long typeouts can be aborted using the rubout key during the list process.

7. The M (Move) Command. The M command allows block movement of program or data areas from one location to another in memory. The form is

Ms,f,d

where s is the start address of the move, f is the final address of the move, and d is the destination address. Data is first moved from s to d, and both addresses are incremented. If s exceeds f then the move operation stops, otherwise the move operation is repeated.

8. The R (Read) Command. The R command is used in conjunction with the I command to read COM and HEX files from the diskette into the transient program area in preparation for the debug run. The forms are

R
Rb

where b is an optional bias address which is added to each program or data address as it is loaded. The load operation must not overwrite any of the system parameters from 000H through 0FFH (i.e., the first page of memory). If b is omitted, then b=0000 is assumed. The R command requires a previous I command, specifying the name of a HEX or COM file. The load address for each record is obtained from each individual HEX record, while an assumed load address of 100H is taken for COM files. Note that any number of R commands can be issued following the I command to re-read the program under test,

assuming the tested program does not destroy the default area at 5CH. Further, any file specified with the filetype "COM" is assumed to contain machine code in pure binary form (created with the LOAD or SAVE command), and all others are assumed to contain machine code in Intel hex format (produced, for example, with the ASM command).

Recall that the command

DDT filename.filetype

which initiates the DDT program is equivalent to the commands

```
DDT
-Ifilename.filetype
-R
```

Whenever the R command is issued, DDT responds with either the error indicator "?" (file cannot be opened, or a checksum error occurred in a HEX file), or with a load message taking the form

```
NEXT PC
nnnn pppp
```

where nnnn is the next address following the loaded program, and pppp is the assumed program counter (100H for COM files, or taken from the last record if a HEX file is specified).

9. The S (Set) Command. The S command allows memory locations to be examined and optionally altered. The form of the command is

Ss

where s is the hexadecimal starting address for examination and alteration of memory. DDT responds with a numeric prompt, giving the memory location, along with the data currently held in the memory location. If the operator types a carriage return, then the data is not altered. If a byte value is typed, then the value is stored at the prompted address. In either case, DDT continues to prompt with successive addresses and values until either a period (.) is typed by the operator, or an invalid input value is detected.

10. The T (Trace) Command. The T command allows selective tracing of program execution for 1 to 65535 program steps. The forms are

```
T
Tn
```

In the first case, the CPU state is displayed, and the next program step is executed. The program terminates immediately, with the termination address

displayed as

*hhhh

where hhhh is the next address to execute. The display address (used in the D command) is set to the value of H and L, and the list address (used in the L command) is set to hhhh. The CPU state at program termination can then be examined using the X command.

The second form of the T command is similar to the first, except that execution is traced for n steps (n is a hexadecimal value) before a program breakpoint occurs. A breakpoint can be forced in the trace mode by typing a rubout character. The CPU state is displayed before each program step is taken in trace mode. The format of the display is the same as described in the X command.

Note that program tracing is discontinued at the interface to CP/M, and resumes after return from CP/M to the program under test. Thus, CP/M functions which access I/O devices, such as the diskette drive, run in real-time, avoiding I/O timing problems. Programs running in trace mode execute approximately 500 times slower than real time since DDT gets control after each user instruction is executed. Interrupt processing routines can be traced, but it must be noted that commands which use the breakpoint facility (G, T, and U) accomplish the break using a RST 7 instruction, which means that the tested program cannot use this interrupt location. Further, the trace mode always runs the tested program with interrupts enabled, which may cause problems if asynchronous interrupts are received during tracing.

Note also that the operator should use the rubout key to get control back to DDT during trace, rather than executing a RST 7, in order to ensure that the trace for the current instruction is completed before interruption.

11. The U (Untrace) Command. The U command is identical to the T command except that intermediate program steps are not displayed. The untrace mode allows from 1 to 65535 (0FFFFH) steps to be executed in monitored mode, and is used principally to retain control of an executing program while it reaches steady state conditions. All conditions of the T command apply to the U command.

12. The X (Examine) Command. The X command allows selective display and alteration of the current CPU state for the program under test. The forms are

X
Xr

where r is one of the 8080 CPU registers

C	Carry Flag	(0/1)
Z	Zero Flag	(0/1)

M	Minus Flag	(0/1)
E	Even Parity Flag	(0/1)
I	Interdigit Carry	(0/1)
A	Accumulator	(0-FF)
B	BC register pair	(0-FFFF)
D	DE register pair	(0-FFFF)
H	HL register pair	(0-FFFF)
S	Stack Pointer	(0-FFFF)
P	Program Counter	(0-FFFF)

In the first case, the CPU register state is displayed in the format

CfZfMfEfIf A=bb B=dddd D=dddd H=dddd S=dddd P=dddd inst

where f is a 0 or 1 flag value, bb is a byte value, and dddd is a double byte quantity corresponding to the register pair. The "inst" field contains the disassembled instruction which occurs at the location addressed by the CPU state's program counter.

The second form allows display and optional alteration of register values, where r is one of the registers given above (C, Z, M, E, I, A, B, D, H, S, or P). In each case, the flag or register value is first displayed at the console. The DDT program then accepts input from the console. If a carriage return is typed, then the flag or register value is not altered. If a value in the proper range is typed, then the flag or register value is altered. Note that BC, DE, and HL are displayed as register pairs. Thus, the operator types the entire register pair when B, C, or the BC pair is altered.

III. IMPLEMENTATION NOTES.

The organization of DDT allows certain non-essential portions to be overlayed in order to gain a larger transient program area for debugging large programs. The DDT program consists of two parts: the DDT nucleus and the assembler/disassembler module. The DDT nucleus is loaded over the Console Command Processor, and, although loaded with the DDT nucleus, the assembler/disassembler is overlayable unless used to assemble or disassemble.

In particular, the BDOS address at location 6H (address field of the JMP instruction at location 5H) is modified by DDT to address the base location of the DDT nucleus which, in turn, contains a JMP instruction to the BDOS. Thus, programs which use this address field to size memory see the logical end of memory at the base of the DDT nucleus rather than the base of the BDOS.

The assembler/disassembler module resides directly below the DDT nucleus in the transient program area. If the A, L, T, or X commands are used during the debugging process then the DDT program again alters the address field at 6H to include this module, thus further reducing the logical end of memory. If a program loads beyond the beginning of the assembler/disassembler module, the A and L commands are lost (their use produces a "?" in response), and the

trace and display (T and X) commands list the "inst" field of the display in hexadecimal, rather than as a decoded instruction.

IV. AN EXAMPLE.

The following example shows an edit, assemble, and debug for a simple program which reads a set of data values and determines the largest value in the set. The largest value is taken from the vector, and stored into "LARGE" at the termination of the program

```

ED SCAN.ASM
*1, 1-1 ORG 1-1 100H ;L, START OF TRANSIENT AREA,
      MVI B, LEN ;LENGTH OF VECTOR TO SCAN,
      MVI C, 0 ;LARGER-RST VALUE SO FAR,
LOOP:  LXI H, VECT ;BASE OF VECTOR,
      MOV A, M ;GET VALUE,
      SUB C ;LARGER VALUE IN C?,
      JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND,
      NEW LARGEST VALUE, STORE IT TO C,
      MOV C, A,
NFOUND: INX H ;TO NEXT ELEMENT,
        DCR B ;MORE TO SCAN?,
        JNZ LOOP ;FOR ANOTHER,

;
; END OF SCAN, STORE C,
      MOV A, C ;GET LARGEST VALUE,
      STA LARGE,
      JMP 0 ;REBOOT,

;
; TEST DATA
VECT:  DB 2, 0, 4, 3, 5, 6, 1, 5,
LEN:    EQU $-VECT ;LENGTH,
LARGE:  DS 1 ;LARGEST VALUE ON EXIT,
END,

*BOF,
ORG 100H ;START OF TRANSIENT AREA
MVI B, LEN ;LENGTH OF VECTOR TO SCAN
MVI C, 0 ;LARGER VALUE SO FAR
LXI H, VECT ;BASE OF VECTOR
LOOP: MOV A, M ;GET VALUE
      SUB C ;LARGER VALUE IN C?
      JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND
      NEW LARGEST VALUE, STORE IT TO C
      MOV C, A
NFOUND: INX H ;TO NEXT ELEMENT
        DCR B ;MORE TO SCAN?
        JNZ LOOP ;FOR ANOTHER

```

Create Source
Program - underlined
characters typed
by programmer.
";" represents carriage
return.

```

;      END OF SCAN, STORE C
MOV    A,C      ;GET LARGEST VALUE
STA    LARGE
JMP     0        ;REBOOT

;
;      TEST DATA
VECT:   DB      2,0,4,3,5,6,1,5
LEN     EQU     $-VECT ;LENGTH
LARGE:  DS      1      ;LARGEST VALUE ON EXIT
END
*E, ← End of Edit

```

ASM SCAN, Start Assembler
CP/M ASSEMBLER - VER 1.0

0122
002H USE FACTOR
END OF ASSEMBLY

Assembly Complete - Look at Program Listing

TYPE SCAN.PRN,

Code Address

Machine Code

Source Program

```

0100 0608      ORG    100H      ;START OF TRANSIENT AREA
0102 0E00      MVI     B,LEN     ;LENGTH OF VECTOR TO SCAN
0104 211901    MVI     C,0       ;LARGEST VALUE SO FAR
0107 7E        LXI     H,VECT    ;BASE OF VECTOR
0108 91        MOV     A,M       ;GET VALUE
0109 D20D01    SUB     C         ;LARGER VALUE IN C?
                                JNC     NFOUND ;JUMP IF LARGER VALUE NOT FOUND
                                ;      NEW LARGEST VALUE, STORE IT TO C
010C 4F        MOV     C,A
010D 23        NFOUND: INX     H   ;TO NEXT ELEMENT
010E 05        DCR     B         ;MORE TO SCAN?
010F C20701    JNZ     LOOP      ;FOR ANOTHER

;
;      END OF SCAN, STORE C
0112 79        MOV     A,C       ;GET LARGEST VALUE
0113 322101    STA     LARGE
0116 C30000    JMP     0         ;REBOOT

;
;      TEST DATA
0119 0200040305VECT: DB      2,0,4,3,5,6,1,5
0008 =         EQU     $-VECT ;LENGTH
0121           DS      1       ;LARGEST VALUE ON EXIT
0122           END
A>

```

Code/data listing
truncated

Value of
Equate

DDT SCAN.HEX,

Start Debugger using hex format machine code

16K DDT VER 1.0

NEXT PC

0121 0000

-X,

last load address +1

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0000 OUT 7F

next instruction
to execute at
PC=0

-XP,

Examine registers before debug run

P=0000 100,

Change PC to 100

-X,

Look at registers again

PC changed.

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,08

-L100,

Next instruction
to execute at PC=100

0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JNC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C

Disassembled Machine
Code at 100H
(See Source Listing
for comparison)

-L,

0113 STA 0121
0116 JMP 0000
0119 STAX B
011A NOP
011B INR B
011C INX B
011D DCR B
011E MVI B,01
0120 DCR B
0121 LXI D,2200
0124 LXI H,0200

A little more
machine code
(note that program
ends at location 116
with a JMP to 0000)

-A116,

enter inline assembly mode to change the JMP to 0000 into a RST 7, which
will cause the program under test to return to DDT if 116H
is ever executed.

0116 RST 7,

0117, (single carriage return stops assemble mode)

-L113, List code at 113H to check that RST 7 was properly inserted

0113 STA 0121
0116 RST 07

In place of JMP


```

0117  NOP
0118  NOP
0119  STAX B
011A  NOP
011B  INR  B
011C  INX  B

```

-X, Look at registers

```

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,00

```

-I, Execute Program for one step. initial CPU state, before \downarrow is executed

```

C0Z0M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MVI B,00*0102

```

-I, Trace one step again (note 08H in B) automatic breakpoint \uparrow

```

C0Z0M0E010 A=00 B=0800 D=0000 H=0000 S=0100 P=0102 MVI C,00*0104

```

-I, Trace again (Register C is cleared)

```

C0Z0M0E010 A=00 B=0800 D=0000 H=0000 S=0100 P=0104 LXI H,0119*0107

```

-I3, Trace three steps

```

C0Z0M0E010 A=00 B=0800 D=0000 H=0119 S=0100 P=0107 MOV A,M

```

```

C0Z0M0E010 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C

```

```

C0Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JNC 010D*010D

```

-D119, Display memory starting at 119H.

Address	02	00	04	03	05	06	01	Program data
0119	02	00	04	03	05	06	01	
0120	05	11	00	22	21	00	02	7E EB 77 13 23 EB 0B 78 B1
0130	C2	27	01	C3	03	29	00	00 00 00 00 00 00 00 00
0140	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0150	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0160	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0170	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0180	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
0190	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
01A0	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
01B0	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00
01C0	00	00	00	00	00	00	00	00 00 00 00 00 00 00 00

automatic breakpoint at 10DH \rightarrow (X) lowercase x

Data is displayed in ASCII with a "0" in the position of non-graphic characters.

-X, Current CPU state \downarrow

```

C0Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H

```

-T5, Trace 5 steps from current CPU state

```

C0Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010D INX H

```

```

C0Z0M0E011 A=02 B=0800 D=0000 H=011A S=0100 P=010E DCR B

```

```

C0Z0M0E011 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107

```

```

C0Z0M0E011 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M

```

```

C0Z0M0E011 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C*0109

```

-U5, Trace without listing intermediate states

```

C0Z1M0E111 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D*0108

```

-X, CPU state at end of U5,

```

C0Z0M0E111 A=04 B=0600 D=0000 H=011B S=0100 P=0108 SUB C

```

-G, Run Program from current PC until completion (in real-time)

*0116 breakpoint at 116H, caused by executing RST 7 in machine code

-X, CPU state at end of Program

C0Z1M0E111 A=00 B=0000 D=0000 H=0121 S=0100 P=0116 RST 07

-XP, examine and change Program Counter

P=0116 100,

-X,

C0Z1M0E111 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08

-T10, Trace 10 (hexadecimal) steps

C0Z1M0E111 A=00 B=0000 D=0000 H=0121 S=0100 P=0100 MVI B,08
C0Z1M0E111 A=00 B=0000 D=0000 H=0121 S=0100 P=0102 MVI C,00
C0Z1M0E111 A=00 B=0000 D=0000 H=0121 S=0100 P=0104 LXI H,0119
C0Z1M0E111 A=00 B=0000 D=0000 H=0119 S=0100 P=0107 MOV A,M
C0Z1M0E111 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C
C0Z0M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JNC 010D
C0Z0M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=010D INX H
C0Z0M0E011 A=02 B=0000 D=0000 H=011A S=0100 P=010E DCR B
C0Z0M0E011 A=02 B=0700 D=0000 H=011A S=0100 P=010F JNZ 0107
C0Z0M0E011 A=02 B=0700 D=0000 H=011A S=0100 P=0107 MOV A,M
C0Z0M0E011 A=00 B=0700 D=0000 H=011A S=0100 P=0108 SUB C
C0Z1M0E111 A=00 B=0700 D=0000 H=011A S=0100 P=0109 JNC 010D
C0Z1M0E111 A=00 B=0700 D=0000 H=011A S=0100 P=010D INX H
C0Z1M0E111 A=00 B=0700 D=0000 H=011B S=0100 P=010E DCR B
C0Z0M0E111 A=00 B=0600 D=0000 H=011B S=0100 P=010F JNZ 0107
C0Z0M0E111 A=00 B=0600 D=0000 H=011B S=0100 P=0107 MOV A,M*0108

-A109, Insert a "hot patch" into the machine code to change the JNC to JC

-G0, Stop DDT so that a version of the patched program can be saved

SAVE 1 SCAN.COM, Program resides on first page, so save 1 page.

A>DDT SCAN.COM, Restart DDT with the saved memory image to continue testing

16K DDT VER 1.0

NEXT PC

0200 0100

-L100, List some code

0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D

Previous patch is present in X.COM

```

010C  MOV  C,A
010D  INX  H
010E  DCR  B
010F  JNZ  0107
0112  MOV  A,C

```

-X,

P=0100,

-T10, Trace to see how patched version operates

Data is moved from A to C

```

C020M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0100 MYI B,00
C020M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0102 MYI C,00
C020M0E010 A=00 B=0000 D=0000 H=0000 S=0100 P=0104 LXI H,0119
C020M0E010 A=00 B=0000 D=0000 H=0119 S=0100 P=0107 MOV A,M
C020M0E010 A=02 B=0000 D=0000 H=0119 S=0100 P=0108 SUB C
C020M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=0109 JC 010D
C020M0E011 A=02 B=0000 D=0000 H=0119 S=0100 P=010C MOV C,A
C020M0E011 A=02 B=0002 D=0000 H=0119 S=0100 P=010D INX H
C020M0E011 A=02 B=0002 D=0000 H=011A S=0100 P=010E DCR B
C020M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107
C020M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M
C020M0E011 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C
C120M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D
C120M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H
C120M1E010 A=FE B=0702 D=0000 H=011B S=0100 P=010E DCR B
C120M0E111 A=FE B=0602 D=0000 H=011B S=0100 P=010F JNZ 0107*0107

```

-X,

breakpoint after 16 steps

```
C120M0E111 A=FE B=0602 D=0000 H=011B S=0100 P=0107 MOV A,M
```

-G,108, Run from current PC and breakpoint at 108H

*0108

-X,

next data item

```
C120M0E111 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C
```

-T,

Single step for a few cycles

```
C120M0E111 A=04 B=0602 D=0000 H=011B S=0100 P=0108 SUB C*0109
```

-T,

```
C020M0E011 A=02 B=0602 D=0000 H=011B S=0100 P=0109 JC 010D*010C
```

-X,

```
C020M0E011 A=02 B=0602 D=0000 H=011B S=0100 P=010C MOV C,A
```

-G, Run to completion

*0116

-X,

```
C021M0E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0116 RST 07
```

-S121, look at the value of "LARGE"

0121 03, Wrong Value!

0122 00,

0123 22,

0124 21,

0125 00,

0126 02,

0127 7E -,

End of the S Command

-L100,

0100 MVI B,08
0102 MVI C,00
0104 LXI H,0119
0107 MOV A,M
0108 SUB C
0109 JC 010D
010C MOV C,A
010D INX H
010E DCR B
010F JNZ 0107
0112 MOV A,C

Review the Code

-L,

0113 STA 0121
0116 RST 07
0117 NOP
0118 NOP
0119 STAX B
011A NOP
011B INR B
011C INX B
011D DCR B
011E MVI B,01
0120 DCR B

-XP,

P=0116 100, Reset the PC

-I, Single step, and watch data values

C0Z1M0E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0100 MVI B,08*0102

-I,

C0Z1M0E111 A=03 B=0003 D=0000 H=0121 S=0100 P=0102 MVI C,00*0104

-I,

C0Z1M0E111 A=03 B=0000 D=0000 H=0121 S=0100 P=0104 LXI H,0119*0107

-I,

C0Z1M0E111 A=03 B=0000 D=0000 H=0119 S=0100 P=0107 MOV A,M*0108

Count set
largest set

base address of data set

-T,

first data item brought to A

C0Z1M0E111 A=02 B=0800 D=0000 H=0119 S=0100 P=0108 SUB C*0109

-T,

C0Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=0109 JC 010D*010C

-T,

C0Z0M0E011 A=02 B=0800 D=0000 H=0119 S=0100 P=010C MOV C,A*010D

-T,

first data item moved to C correctly

C0Z0M0E011 A=02 B=0802 D=0000 H=0119 S=0100 P=010D INX H*010E

-T,

C0Z0M0E011 A=02 B=0802 D=0000 H=011A S=0100 P=010E DCR B*010F

-T,

C0Z0M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=010F JNZ 0107*0107

-T,

C0Z0M0E011 A=02 B=0702 D=0000 H=011A S=0100 P=0107 MOV A,M*0108

-T,

second data item brought to A

C0Z0M0E011 A=00 B=0702 D=0000 H=011A S=0100 P=0108 SUB C*0109

-T,

subtract destroys data value which was loaded!!!

C1Z0M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=0109 JC 010D*010D

-T,

C1Z0M1E010 A=FE B=0702 D=0000 H=011A S=0100 P=010D INX H*010E

-L100,

0100 MVI B,08

0102 MVI C,00

0104 LXI H,0119

0107 MOV A,M

0108 SUB C

0109 JC 010D

010C MOV C,A

010D INX H

010E DCR B

010F JNZ 0107

0112 MOV A,C

-A108,

0108 CMP C hot patch at 108H changes SUB to CMP

0109,

-G0, stop DDT for SAVE

← This should have been a CMP so that register A would not be destroyed.

SAVE 1 SCAN.COM

Save memory image

A>DDT SCAN.COM

Restart DDT

16K DDT VER 1.0

NEXT PC

0200 0100

-XP

P=0100

-L116

0116 RST 07

0117 NOP

0118 NOP

0119 STAX B

011A NOP

- (rubout)

} Look at code to see if it was properly loaded
(long timeout aborted with rubout)

-G.116 Run from loop to completion

*0116

-XC Look at Carry (accidental typo)

C1

-X Look at CPU state

C1Z1M0E111 A=06 B=0006 D=0000 H=0121 S=0100 P=0116 RST 07

-S121 Look at "Large" - it appears to be correct.

0121 06

0122 00

0123 22

-G0 stop DDT

ED SCAN.ASM

Re-edit the source program, and make both changes

*NSUB

*0LT

SUB C
*SSUBZCMPZ0LT
CMP C

;LARGER VALUE IN C?

;LARGER VALUE IN C?

*

JNC NFOUND ;JUMP IF LARGER VALUE NOT FOUND

*SNCZCZ0LT
JC

NFOUND ;JUMP IF LARGER VALUE NOT FOUND

*E

ASM SCAN.AAZ, Re-assemble, selecting source from disk A
 hex to disk A
 Print to Z (selects no print file)

0122
 002H USE FACTOR
 END OF ASSEMBLY

DDT SCAN.HEX, Re-run debugger to check changes

16K DDT VER 1.0
 NEXT PC
 0121 0000
 -L116,

0116 JMP 0000 check to ensure end is still at 116H
 0119 STAX B
 011A NOP
 011B INR B
 - (rubout)

-G100,116, Go from beginning with breakpoint at end

*0116 breakpoint reached

-D121, Look at "LARGE" correct value computed

0121 06 00 22 21 00 02 7E EB 77 13 23 EB 08 78 B1 .. "I...W.#...X.
 0130 C2 27 01 C3 03 29 00 00 00 00 00 00 00 00 00).....
 0140 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

- (rubout) aborts long typeout

-GO, stop DDT, debug session complete

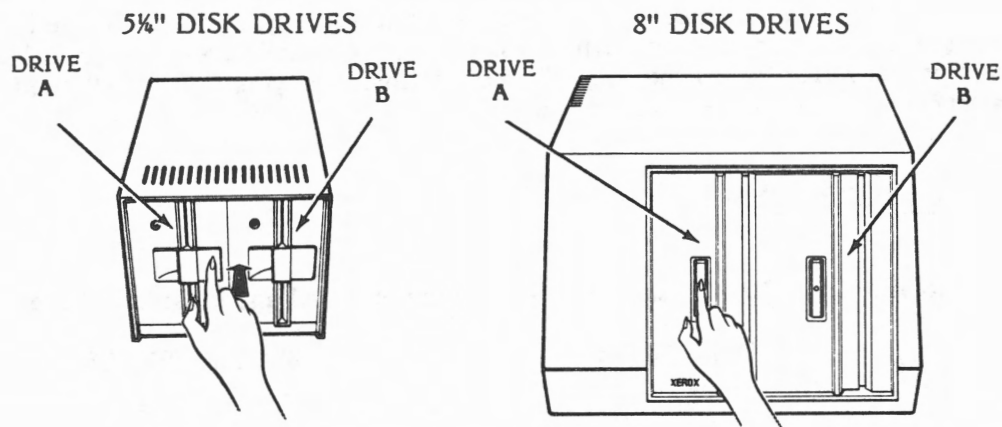
DIAGNOSTICS

Your 820 comes with a Diagnostic Exerciser disk. You can use this disk to check that your 820 is in proper working order. The disk will check the different components of the system and display a message if it finds a malfunction.

It's a good idea to check your system when you first install it. If you experience any problems while using the system, you can use the Diagnostic Exerciser to check the system and find out if it needs to be serviced (repaired).

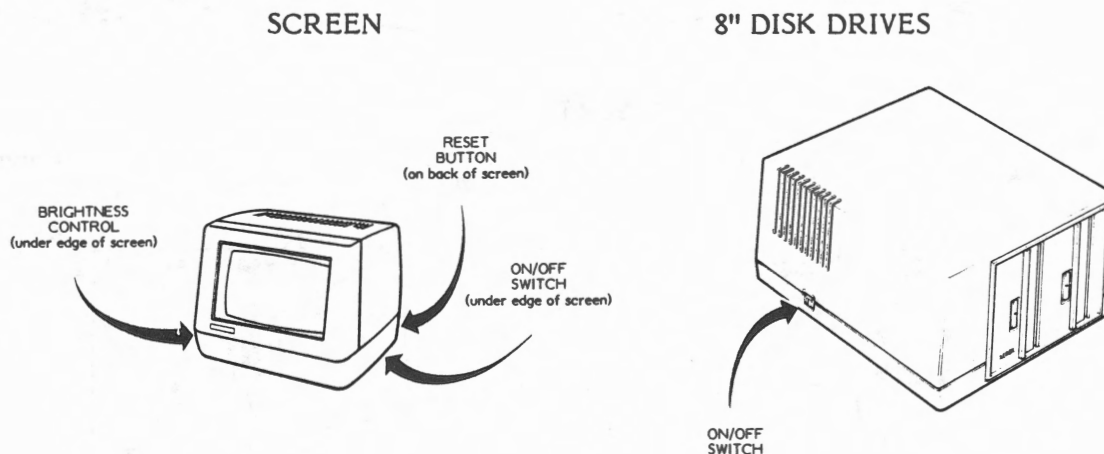
HOW TO BEGIN:

OPEN the disk drives (shown below) and remove any disks



LOCATE the ON/OFF switch on the left side of the 8" disk drives (shown below)
(The 5 1/4" disk drives does not have a ON/OFF switch)

LOCATE the ON/OFF switch under the right side of the screen (shown below)



TURN the screen on -- if it is already turned on, press the RESET button at the back of the screen

TURN the 8" disk drives on -- if your system has 8" drives, they must be turned on for your system to operate.

WAIT for the word XEROX to appear on the screen. If it does not appear after a few seconds, try adjusting the brightness control (under the left edge of screen)

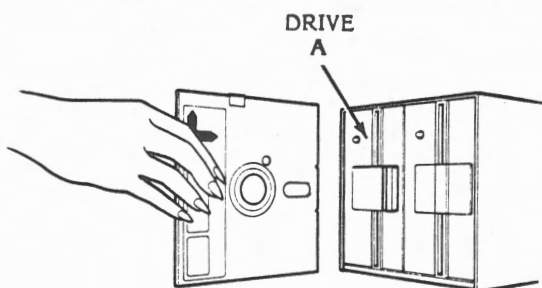
If nothing appears on the screen, your system needs servicing

When you use the Diagnostic disk to check out your system, you'll need to have an initialized disk to put in the other disk drive. (An initialized disk is a disk that has been prepared for use in the 820.) If you just installed your 820, you'll need to use the instructions below to initialize a disk.

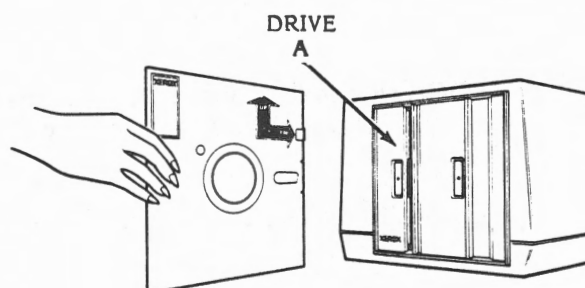
TO INITIALIZE A DISK:

INSERT the CP/M disk in the left drive (Drive A) and close the drive

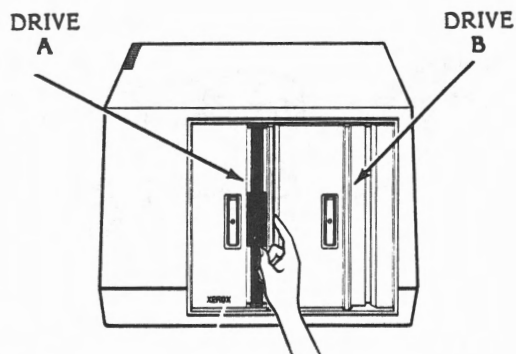
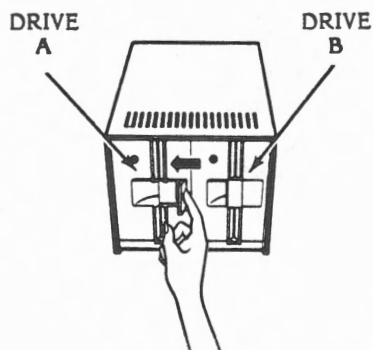
5 1/4" DISK DRIVES



8" DISK DRIVES



INSERT DISK

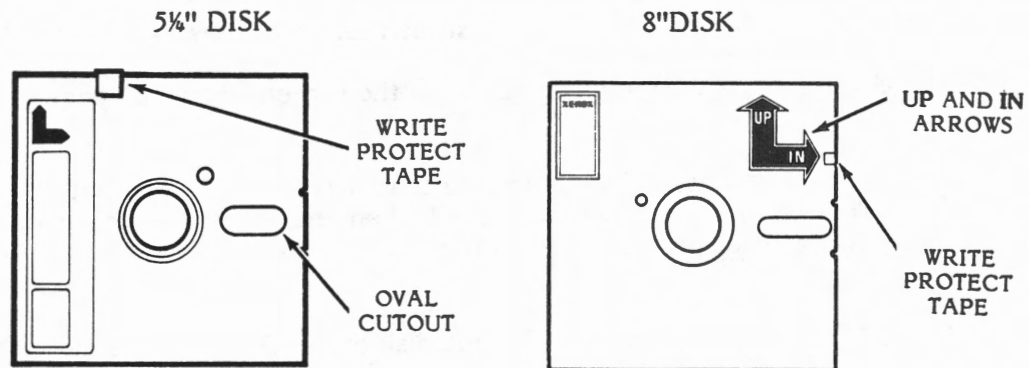


CLOSE DRIVE

DIAGNOSTICS

TYPE A and press RETURN

OBTAIN at least one new disk and take the write protect tape off of the 5 $\frac{1}{4}$ " disk, or put a write protect tape on the 8" disk.
(If you don't have a new disk, a previously used one will do.)



TYPE the word **INIT** and press RETURN

WAIT for the message "ENTER DISK DRIVE TO BE INITIALIZED (A or B)"

INSERT the new disk in drive B

TYPE B and press RETURN twice

WAIT for the message "0 FLAWED SECTORS" to display

If the disk has flawed sectors indicated by a number other than 0 in front of the FLAWED SECTORS message, do not use it - initialize another disk using the steps below.

- Replace the disk in drive B with another disk
- Type B and press RETURN twice
- Wait for the 0 FLAWED SECTORS message to appear

REMOVE both disks

You're now ready to check out the system with the Diagnostic Exerciser disk. The instructions for running diagnostics are on the next page.

TO RUN DIAGNOSTICS:

Before you run diagnostics, you'll need to put paper in your printer (use two sheets, so that the paper covers the width of the platen), and turn on the printer. The On/Off Switch is on the back of the Diablo 630 printer.

As the test runs, you'll need to watch the screen for error messages.

PRESS the RESET button on the back on the screen (shown on page 9-1)

CHECK the Diagnostic disk to be sure that it is not write protected.
(The 5 $\frac{1}{4}$ " disks are not protected when the tapes are removed, and the 8" disks are not protected when the tapes are on.)

INSERT the Diagnostic disk in the left disk drive (Drive A) and close the drive

INSERT an initialized disk in the right disk drive (Drive B) and close the drive
(Be sure that the disk is not write protected)

PRESS the A key and the RETURN key

The diagnostic disk will begin to check out the system. Be careful not to touch any keys on the keyboard while the test is running. Touching a key can stop the test.

Note: If there is an error during the test, pressing **CTRL + C** will run the remaining tests

WATCH the screen for the results of the first memory test. In about 30 seconds, the screen should show:

PASSES COMPLETE = 0001 ; COUNT OF ERROR BYTES = 0000
PASSES COMPLETE = 0001 ; COUNT OF ERROR BYTES = 0000

If the COUNT OF ERROR BYTES does not equal 0000, your system needs to be serviced.

WATCH the screen for the results of the second memory test. The screen should show:

PASSES COMPLETE = 0001 ; COUNT OF ERROR BYTES = 0000
PASSES COMPLETE = 0001 ; COUNT OF ERROR BYTES = 0000

If the COUNT OF ERROR BYTES does not equal 0000, your system needs to be serviced.

WATCH your screen for the results of the disk test. The disk drives will click during this test. When finished the screen should show:

0 read/write error detected
0 seek errors detected

WATCH the screen test as it displays the screen test. The test pattern should fill the screen with characters. (The boader around the test pattern will remain black.)

If there are missing characters or irregularities in the test pattern, your system needs to be serviced.

(If you do not have a printer, the test will end here. Remove both disks and press the RESET button.)

WAIT while the printer prints its test pattern. (If you have an 88 or 92 character wheel on the 630 Printer, the test pattern will have blanks in some places.)

If the test pattern did not print the alphabet and numbers, or if the characters were not properly aligned on the paper, your system may need to be serviced.

LOOK for the message "DIAGNOSTIC COMPLETE. RESET TO CONTINUE"

REMOVE the Diagnostic disk and the initialized disk, and place them back in their disk envelopes. Your manual has a disk holder at the back that you can use for storing your Diagnostic Exerciser disk.

PRESS the RESET button.

If all the test messages corresponded with those shown in the instructions, your 820 is hooked up correctly and in proper working order.

(This page intentionally left blank)

DIABLO 630 PRINTER

INTRODUCTION

Diablo Systems has combined the simplicity and reliability of a newly designed daisy wheel printer with the latest microelectronic technology to produce the Model 630 Printer.

The Model 630 is a universal RS 232-C interface printer. It will support a heavy work load using conventional serial data interchange techniques and protocols.

The Model 630 can use all Diablo and Xerox plastic and metal print wheels.

The Model 630 can use many of Diablo's present paper handling devices, such as forms tractors, sheet feeders, etc.

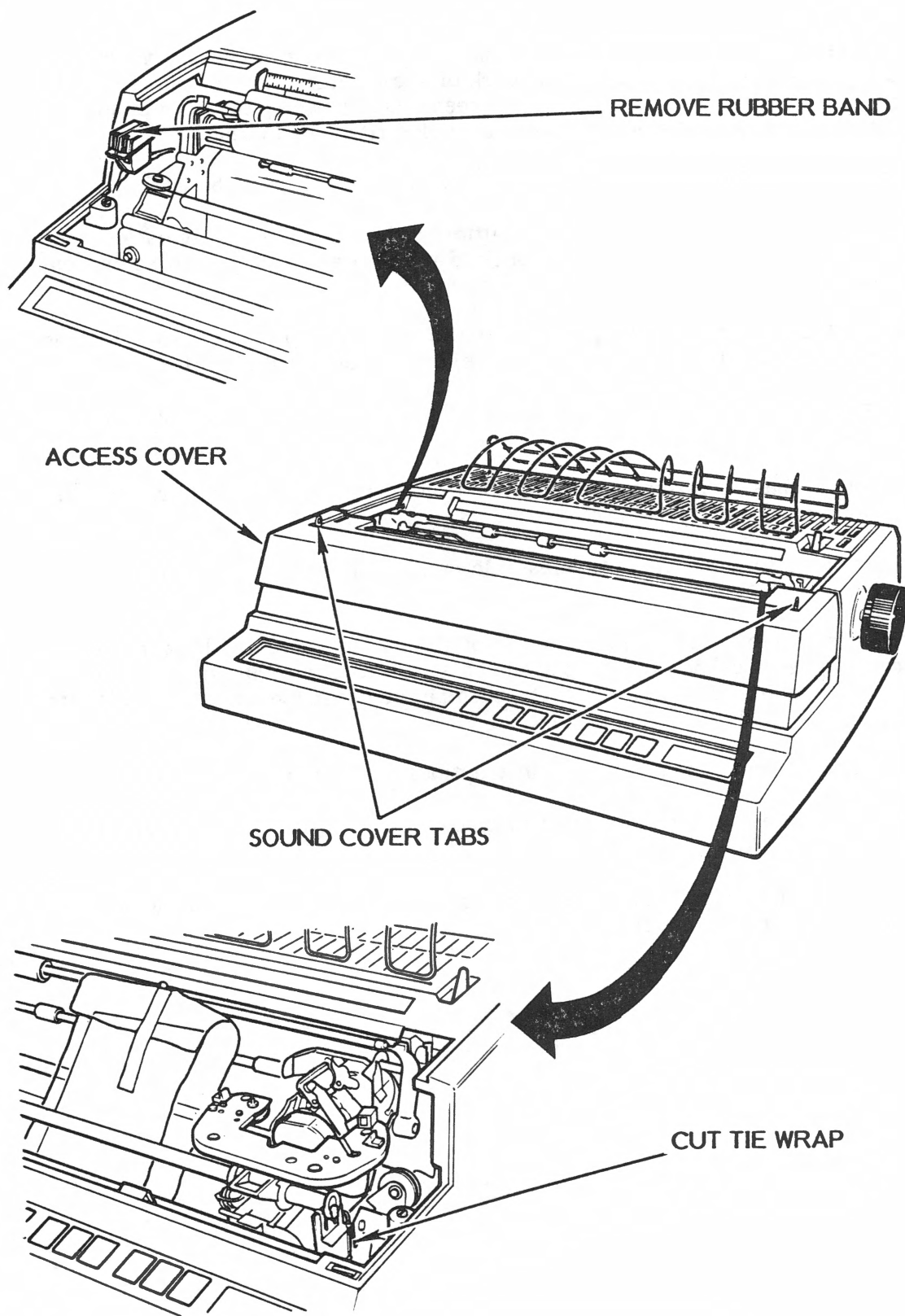
UNPACKING THE 630

1. Take the printer and all accessories out of the carton and remove the plastic dust bag. Place the Model 630 on a desk or table.

NOTE: The weight of the Model 630 is centered toward its right rear (as you look at it); thus it is easiest to lift by holding it at the right-rear and left-front.

2. Inspect the Model 630 and any accompanying accessories for evidence of shipping damage. Immediately notify the shipping agent of any damage to the unit or its parts.
3. Remove the access cover. It is held in place by magnetic latches in front and small tabs in back.
4. Remove the plastic bag and the CAUTION tag attached to the paper bail by cutting or removing the tie wrap (nylon strap).
5. Remove or cut the following shipping restraint items if installed (see pictures on opposite page):
 - a) Rubber band securing the cover open switch actuator.
 - b) Rubber band securing the paper cradle to the paper pressure rollers (if no platen is installed).
 - c) Tie wrap (nylon strap) holding the carriage to the right printer frame.

CAUTION: DO NOT cut any other tie wraps!



REMOVING SHIPPING RESTRAINTS

SYSTEM COMPONENTS

10-3

INSTALLATION

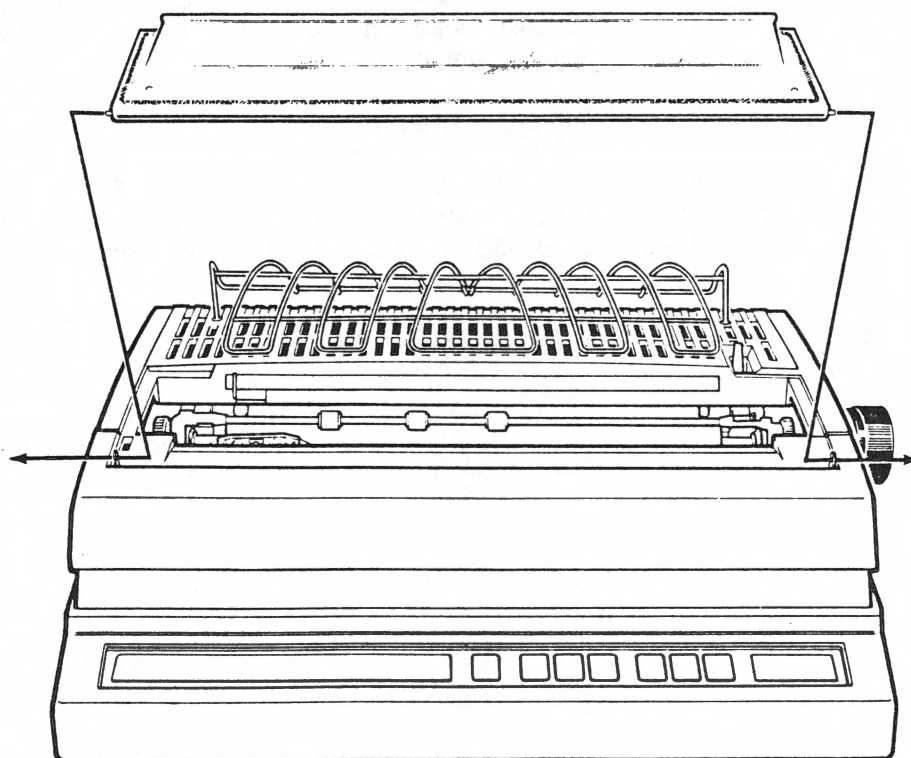
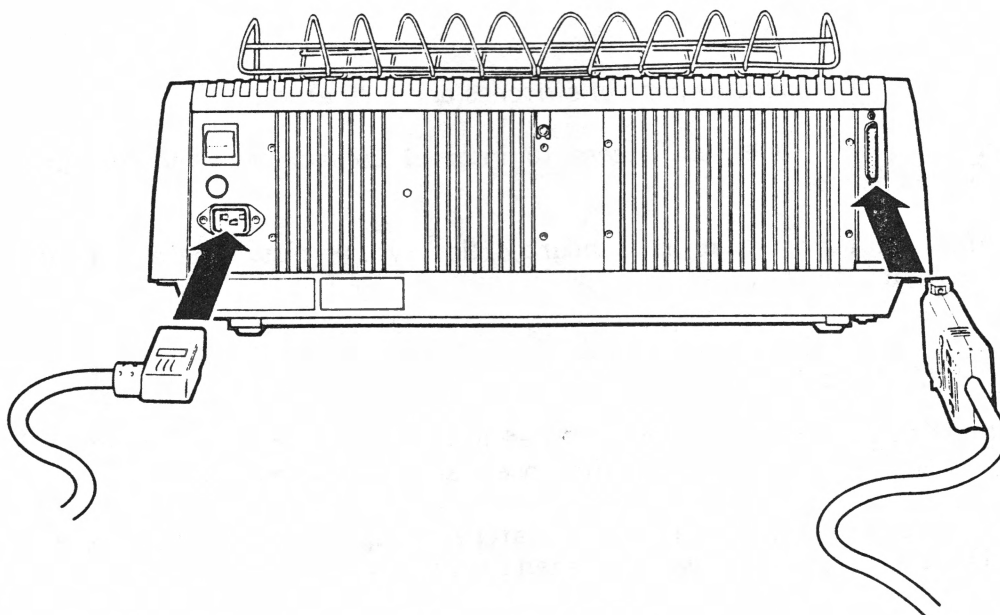
1. Place the Model 630 on your desk or table. Look at the two cords. One printer cord plug fits the large outlet on the back of the printer; the other plug fits the large outlet on the back of the 820's screen. One power plug fits the smaller outlet on the back of the printer; the other plug fits a wall outlet.
2. Check the ON/OFF switch on the back of the printer -- it must be OFF!
3. Plug the printer cord into the large outlet (right side, looking at back -- see picture on the opposite page) on the back of the printer. Tighten the screws on the plug.
4. Plug the L-shaped end of the power cord into the small outlet (left side, looking at back -- see picture on the opposite page) on the back of the printer.

Plug the other end of the power cord into the wall outlet. **DO NOT** turn on the printer yet.

5. If the platen has been removed, reinstall it by lowering it down into place while pressing down on both platen release levers.

NOTE: Make sure the platen's releasable drive gear end is on the right (as you view it).

6. Install the platen knob on the right end of the platen - thru the hole in the top cover. Engage the knob on the platen shaft, rotate the knob until its slot fits over the cross pin on the platen shaft, then push against the knob to snap it into place.
7. Install a print wheel and ribbon (see instructions on Pages 10-8 and 10-9).
8. Install the access cover. It is held in place by magnetic latches in front and small tabs in back.
9. Install the sound cover. Insert the sound cover pivot shaft in the tabs located on the top of the access cover.



SYSTEM COMPONENTS
10-5

FAMILIARIZATION

(See picture on opposite page)

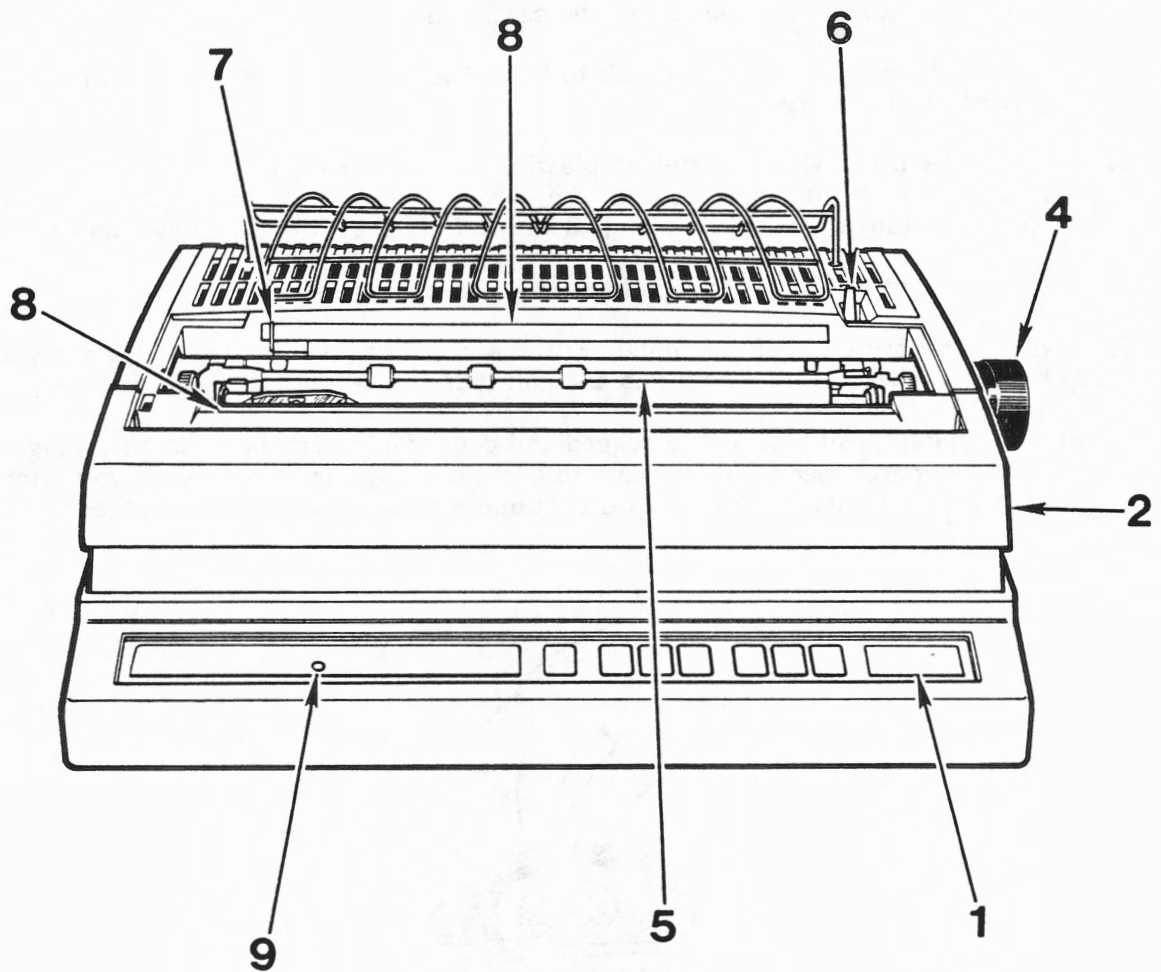
1. Control Panel - Operator control switches and indicators.
2. Access Cover - Operator access to internal controls and for changing print wheels and ribbons.
3. Platen - similar to those on standard office typewriters. There is a hand knob on the right side only.

Paper Handling Features

4. Platen Knob - This knob, when pushed in (to disengage the drive gear), allows the operator to rotate the platen to insert and position paper.

Platen Release Latches - These two latches are pressed down simultaneously to insert a platen or to release the platen for removal.

5. Paper Bail - The paper bail holds the paper against the platen for optimum print quality and quietness. The bail must be pulled forward (away from the platen) when inserting paper.
6. Paper Release Lever - This lever releases paper roller pressure when pulled forward, allowing the paper to be positioned manually. Returning this lever to its back position reestablishes paper roller pressure.
7. Paper Guide - A movable guide for aid in inserting paper.
8. Paper Scales - There are two scales associated with the Model 630. One is a column indicator, located on the top cover; the second is a paper scale mounted on the access cover. These scales aid the operator in centering and spacing paper and copy.
9. Power Indicator Light - This light indicates that the power is on.



630 PRINTER

PREPARATION FOR OPERATION

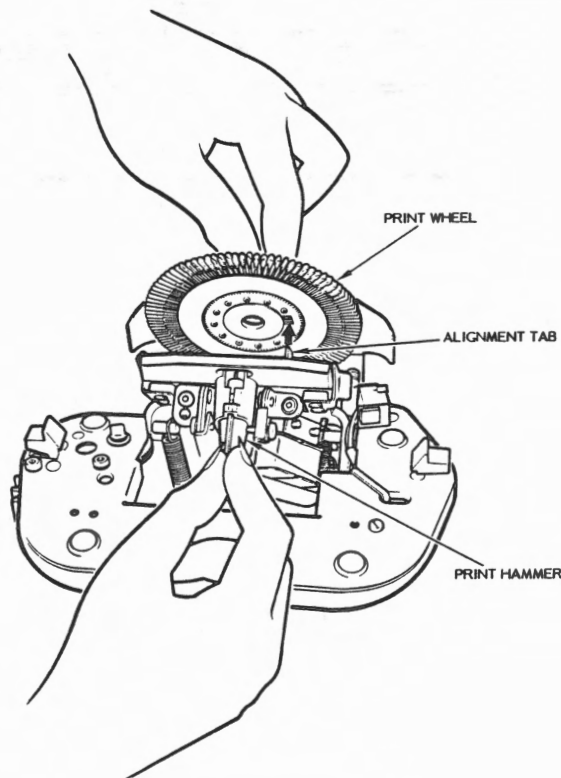
Installing a Print Wheel

CAUTION: Ensure that power to the Model 630 is turned OFF!

1. Grasp the print hammer assembly and pull it toward you to tilt the print wheel mechanism away from the platen and card guide.
2. Rotate the print wheel motor hub to bring the hub's alignment tab to the upper part of its arc of travel.
3. Grasp the print wheel (metal or plastic) by its rubber hub and place it on the print wheel motor hub. Align the wheel's alignment slot with the hub's alignment tab, and push the wheel firmly to fully seat it on the motor hub.
4. Tilt the print wheel mechanism back to its operating position.

Removal of the print wheel is simple. Tilt the print wheel mechanism toward you, grasp the print wheel by its rubber hub and pull it free of the print wheel hub.

Note: Diablo print wheels are rugged and dependable, but they can be damaged. Use care when handling them to avoid bending the "spokes". Always store print wheels in their plastic containers when they are not installed in the printer.



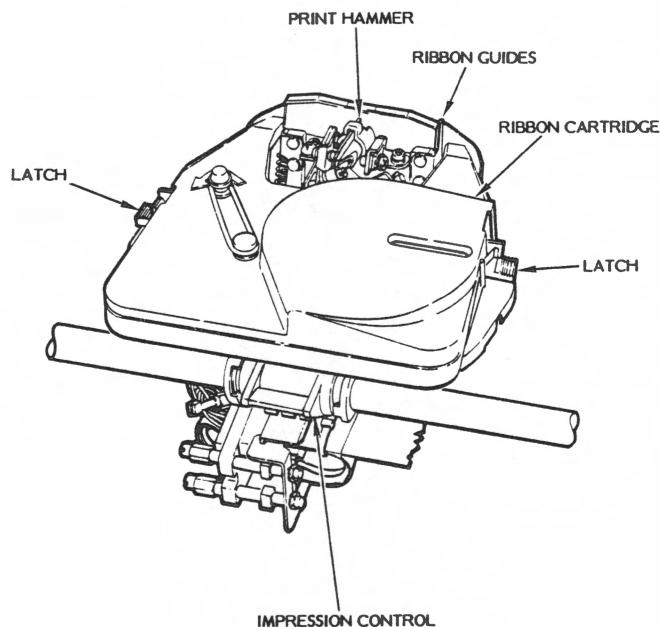
INSTALLING PRINT WHEEL

Installing a Ribbon Cartridge

1. Open the plastic envelope and take out the ribbon cartridge. Note the small knob on the top surface of the cartridge for advancing the ribbon manually. Use this knob to take up any slack in the exposed portion of the ribbon and to make sure the ribbon is tight and straight.
2. Hold the cartridge in one hand with the exposed ribbon toward the platen. Lower the cartridge down over the print hammer guide (orange stripe). Be sure the exposed ribbon is straight and located between the card guide and print wheel. Push the cartridge down firmly until both latches have snapped into position. Rock the cartridge back and forth on the platform to ensure that the ribbon is free to move up and down. Turn the ribbon advance knob a turn or two to ensure that the ribbon is tight, straight and ready to operate.

The ribbon cartridge may be removed by pressing down on both latches simultaneously. The cartridge will be raised up slightly and may be grasped easily and lifted out of the printer.

Note: When a ribbon cartridge is nearly empty, a yellow cross-hatched pattern will appear on the visible back side of the ribbon. The Model 630 will stop printing, sound its alarm and the RIBBON/PAPER light will come on if printing is attempted with the ribbon in the warning (yellow) zone.



INSTALLING A RIBBON

Inserting Paper or Forms

Inserting paper or forms into the Model 630 is accomplished in much the same manner as in a standard typewriter. Paper is inserted down behind the metal paper out bail and platen while the platen is turned manually to bring the paper around and up in front of the platen. The front paper bail (Item 5, page 10-6) aids in guiding the paper back over the platen to the rear when pulled forward. The paper release lever at the right-hand side of the printer may be pulled forward to release roller pressure after paper insertion so the paper can be properly aligned in the printer. After paper is positioned, both the front paper bail and paper release lever are returned to their operating positions.

Paper Thickness/Print Intensity Adjustment

The two-position Multipcopy lever located at the front of the carriage assembly adjusts for paper thickness and print intensity. Setting the lever to its upper position moves the carriage close to the platen, and actuates a switch to the proper setting for light and medium weight paper and form sets of up to two carbon copies. For heavier paper or form sets of up to five copies, the Multipcopy lever is set to its lower position. This rocks the carriage away from the platen slightly, and deactivates the switch to enable an increased print intensity.

To avoid the possibility of ribbon damage, the Multipcopy lever should always be set at its upper position when printing on single sheets of paper using carbon ribbons.

OPERATING THE MODEL 630 PRINTER

Preliminary Steps

Install a print wheel, a ribbon cartridge and insert a sheet of paper - as outlined on pages 10-8, 10-9 and 10-10.

Move the carriage manually to the right a short distance.

Move the power ON/OFF switch at the right rear of the printer to ON.

The POWER indicator should glow; the carriage should move to the left slowly, and then back to the right, to stop at the first print position; and the print wheel should rotate and stop at its "home" position (i.e., the "flag" on metal print wheels should be at the top if the Print Wheel Select switch - under the access cover - has been properly set). This entire process is called the INITIALIZATION, RESET or RESTORE sequence. It clears all volatile memory, resets all position counters and sets the Model 630 to print the first character.

Paper Handling Accessories

-Forms Tractor/Pin Feed Platen: These devices facilitate precision handling of the continuous or manifold paper forms and are provided in both unidirectional and bidirectional versions.

-Mechanical Front Feeder: This device also mounts on top of the Model 630. It is intended for use with difficult, multipart forms and the heavier ledger card stocks.

-Bottom Feed: The Model 630 can be configured for feeding continuous or manifold forms up thru the bottom of the machine. It must be used with either a pin feed platen or a forms tractor.

SPECIAL CONSIDERATIONS

1. Cleaning Print Wheels

Print Wheels used with carbon ribbons seldom need cleaning. Both Plastic Print Wheels and Metal Print Wheels used with cloth ribbons will require an occasional cleaning.

Remove the print wheel and clean with toluene or naphtha* and a soft brush or wiper. DO NOT clean with water. DO NOT get solvent on the hub or damper ring (metal wheels). Be careful not bend the "spokes".

2. Changing Ribbons During Operation

As outlined on page 10-9, the Model 630 will stop printing, the RIBBON/PAPER light will appear, and the alarm will sound upon reaching the end of a carbon ribbon. Should this happen during receipt and print out of data from a host system, the operator should open the access cover, replace the ribbon cartridge as described, close the access cover, and then touch the Control Panel RESET switch to resume printing.

3. Cleaning The Print Hammer

Remove the print wheel and ribbon cartridge as described. Locate the movable print hammer inside its guide, as shown on page 10-8. Push the hammer out to the rear, to expose as much of the head as possible. Use toluene or naphtha* and a wiper or brush to remove any accumulated ink or other substances, and wipe dry.

* Toluene and naphtha are available at most drug stores.

** 90% isopropyl alcohol may be used as a solvent for cleaning the print hammer.

Setting The Switches Under The Access Cover (left to right)

The 630 Printer has been factory preset to the proper switch positions for use with a Xerox 820 Information Processor. For use with other print wheels, the available setting positions are as follows:

1. Print Wheel Select Switch. Set this switch to match the particular type of print wheel being used. This ensures your text will print correctly and prevents possible print wheel damage or excessive wear.

PRINT WHEEL SELECT:

0:	88 Metal
2:	92 Metal
3:	96 Metal
4:	96D Metal
5:	APL Metal
6:	APL Plastic
7:	Plastic
1,8,9:	Optional

2. Spacing Select Switch. This switch selects the horizontal spacing for character printout. Set this switch to 1 for 10 Pitch PWS or 2 for 12 Pitch PWS.

0:	Proportional
1:	10
2:	12
3:	15
4 - 9:	Self Test

For use with the Xerox 820 Information Processor in WP, the switches to the right of Print Wheel and Spacing switches should be positioned toward the front of the printer, except for the BAUD switch marked 120. It should be positioned toward the back of the printer.

Setting The Operating Switches

These six switches are located in the right-hand area of the control panel where they are accessible to the operator with all covers on the machine. These are membrane type momentary action switches actuated by a touch of the finger.

1. RESET Switch. This switch will restore the Model 630 to normal operating status following a printer check or an error condition, and clears all error indicators.
2. SCROLL Switch. Touching this switch advances the paper a small amount to give the operator a clear view of the last printed line. The paper is automatically returned to the last printing position when the switch is released.
3. LF Switch. Touching this switch initiates a single or a double line feed operation, as selected by the DOUBLE L.F. MODE SWITCH. Action is repeated if the switch is held activated longer than 600msec. A line feed code will not be transmitted.
4. FF Switch. Touching this switch initiates a form feed to the next top-of-form position. A form feed code is not transmitted.
5. HERE IS Switch. Touching this switch causes a special "Here Is . ." message of up to 31 characters to be transmitted over the communications link when operating in remote ASCII mode with the fully featured HPRO5 option installed. This is not used with the Xerox 820 Information Processor.
6. BREAK Switch. Touching this switch causes a Break (250msec space) to be transmitted over the communications link when operating in remote mode.
7. (Audio Alarm). This device buzzes briefly to indicate the occurrence of various errors or operating conditions.

Reading The Control Panel Indicators (left to right)

1. POWER. Indicates that AC power is applied to the Model 630.
2. PRINT CHK*. Indicates that a print operation has been called for while the printer is in a "check" condition. A check condition occurs when a print wheel or carriage movement command has been received but cannot be successfully completed due to a malfunction. This condition disables the Model 630 until a restore sequence clears the check condition.

RESET. Note that if the problem causing the check condition has not been corrected when a restore sequence has been initiated, the check will reappear as soon as printing is attempted.

3. PARITY. Indicates detection of any of the following types of error:
 - Incorrect parity sensed on received character.
 - A framing error (no stop bit) detected on a received non-break character.
 - A serial data character detected with an excess number of bits.

When a parity error is detected, a DEL character is substituted for the erroneous character.

This indicator functions only if the PARITY ENABLE switch (under the access cover) is ON.

4. OVFL*. Indicates that the Model 630's print input memory (buffer) is too full (has overflowed). Protocol has not been used properly.
5. RIBBON/PAPER*. Indicates end of ribbon has been reached or that the printer is out of paper, and printing has been attempted.
6. COVER*. Indicates that printing was attempted with the sound cover open.

* - These errors cause a Break to be transmitted when the Model 630 is in Remote mode if DC1/DC3 protocol has not been selected.

SPECIFICATIONS

Print Speed: Up to 40 characters per second with metalized print wheels.

Character Set: 88, 92 or 96 printable characters per print wheel. Switch selectable program support for APL and all ENGLISH language print wheels.

Print Wheels:

Metal	-88 character Xerox
	-96 character Diablo and Xerox
Plastic	-96 character Diablo

Character Spacing: 10 and 12 pitch.

Column Spacing: 1/120 inch (.21mm) minimum.

Print Line: 13.2 inches (335.3mm)
132 columns 10 pitch
158 columns 12 pitch
198 columns 15 pitch

Paper Width: 16.53 inches (419.9mm) maximum
- friction feed without Top Paper Out switch.
16.00 inches (406.4mm) maximum
- friction feed with Top Paper Out switch.
15.25 inches (387.4mm) maximum
full width with optional forms tractor (14.75 inches/-374.7mm between holes). 3.25 inches (82.55mm) minimum with forms tractor (2.75 inches/69.85mm between holes).

Carriage Speed: 400msec maximum for 13.1 inches (332.77mm) of motion.

Tabulation: Left or right.

Line Spacing: 1/48 inch (.53mm) minimum.

Paper Feed: Bidirectional, except with unidirectional forms tractor and uni-directional pin feed platen.

Paper Feed Speed: 4 inches (101.6mm) per second plus 40msec (typical) settling delay time.

Sensors: End of ribbon, top paper out, and cover open.

Paper Thickness:

.000 - .101 inch (.254mm) at low setting (1-3 part forms)
.010 - .027 inch (.254 - .686mm) at high setting (4-6 part form).

Other Features: Self-test; host program control thru escape sequences; data receive/transmit speed selection.

Power Requirements: Strappable for operation from nominal 100, 120, 220 or 240 volt (+10%/-15%) AC inputs, 49-61 Hz. 350W maximum power consumption. Factory preset for 120 VAC.

CHECK YOUR MODEL 630's SERIAL PLATE FOR PROPER INPUT POWER.

This page intentionally left blank.

FORMS TRACTORS

The unidirectional and bidirectional forms tractors are very similar except the unidirectional does not have the reverse drive sprockets required for bidirectional paper feeding.

Unidirectional Forms Tractor - This unit mounts on top of the printer cover where it engages the platen shaft for alignment and drive. It requires use of a friction feed platen. It is adjustable to any paper width from 2-3/4" (69.85mm) to 14-1/2" (368.3mm) maximum as measured between the pin feed drive holes.

Bidirectional Forms Tractor - This unit mounts on top of the printer cover where it engages the platen shaft for alignment and drive. It requires use of a friction feed platen. It is adjustable to any paper width from 2-3/4" (69.85mm) to 14-1/2" (368.3mm) maximum as measured between the pin feed drive holes. It also features both forward and reverse pin feed paper drives to enable feeding paper in either direction.

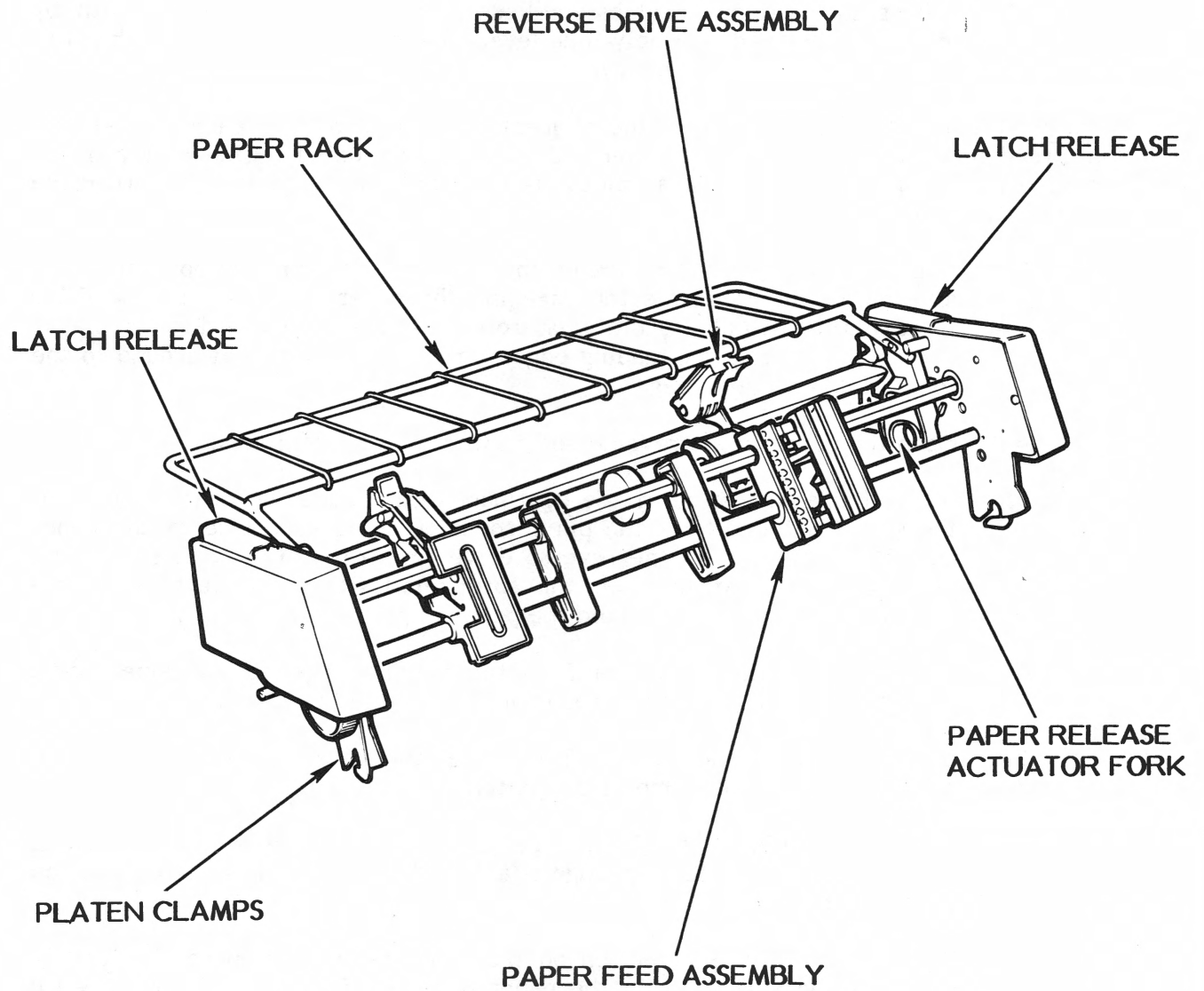
Forms Tractor Installation

Note: If your Model 630 has a sound panel, it must be the special sound panel designed for use with the forms tractors.

- 1) If you have one, swing the sound panel forward to its open position.
- 2) Pull the paper release lever and paper bail forward toward the front of the Model 630 (see Picture, page 10-7).
- 3) Holding the forms tractor at both ends, depress the two latch release levers on the tractor and lower it onto the platen shaft (see Picture, page 10-21). As the tractor is being lowered, guide the paper release actuator fork on the tractor over the paper release lever on the Model 630.
- 4) Release the latch levers to clamp the tractor assembly onto the platen shaft. Check to see that both ends of the tractor are firmly latched.
 - a) Bidirectional forms tractors: Swing the paper support rack forward then backward to make sure the paper release lever stays inside the release actuator fork. If the paper release lever slips out, remove the forms tractor and repeat steps 2 through 4 of this installation procedure.
 - b) Unidirectional forms tractors: The unidirectional tractor is equipped with adjustable supports. Adjust these to hold the back of the tractor to its forward position, and check to see that the paper release lever on the Model 630 snaps in its backward position. Swing the paper support rack back to its normal position; the paper release lever should be pulled to its forward position. This releases the paper tension, which is necessary when operating with the forms tractor.

Forms Tractor Removal

- 1) Remove any paper from the forms tractor.
- 2) Swing the sound panel forward.
- 3) Depress the two latch release levers (see Picture, page 10-21) and lift the tractor straight up.



Loading Paper into the Bidirectional Forms Tractor

To adapt the following procedure to a unidirectional tractor, simply disregard the steps relating solely to the reverse drive assemblies.

- 1) If you have one, swing the sound panel open.
- 2) Adjust the two pin feed assemblies to the appropriate form width by loosening the feed assembly lock levers and sliding the feed assemblies to the left or right as necessary.

Note: The two feed assemblies on the bidirectional forms tractor each consist of a pin feed forward drive belt and a reverse drive sprocket joined together by a metal bracket. The unidirectional forms tractor is equipped only with the forward drive belt assemblies.

- 3) Swing the paper support rack on the tractor to its forward position. This allows convenient access for engaging the paper onto the reverse drive sprockets; it also moves the pressure release lever backward so the platen will be able to grip the leading edge of the paper and pull it around to the forward drive assemblies.
- 4) Open the gates on the reverse and forward drive assemblies.
- 5) Bring the leading edge of the continuous form paper up over the rear of the Model 630 and hook the pin feed holes along each side of the paper onto the feed pins on the reverse drive sprockets. (Be certain that the paper is aligned straight on the sprockets.) Close the gates over the reverse drive sprockets to hold the paper in place on the feed pins.
- 6) Slowly rotate the platen to feed the leading edge of the paper down behind the paper bar, down around and then up in front of the platen.
- 7) Swing the paper rack back. This moves the paper release lever on the printer to release the grip of the platen on the paper.
- 8) Grasp the leading edge of the paper and pull it up while manually turning the platen until the leading edge is above the forward drive assemblies on the tractor.
- 9) While gently creating tension on the paper by pulling up on the leading edge, fit the side holes in the paper onto the feed pins of the forward drive belts. Close the gates to hold the paper in place on the feed pins. Be certain that the paper is aligned straight.
- 10) Move the paper bail back toward the platen. (The bail will be held away from the platen slightly; this is proper when operating with the forms tractor.)
- 11) Rotate the platen 2 or 3 turns forward and backward to check that the paper is feeding properly through the drive assemblies.
- 12) Close the sound panel.

--INTRODUCTION--

The XEROX 820 system monitor is the basic control program for the single-board computer. It begins execution when the computer is first turned on, or whenever the reset button is pressed, and resides in the top 4K of memory.

The monitor provides two essential functions for the system. It is the initial software level of the computer and it contains the routines that initialize and control all the basic system input/output resources. The "front panel" functions of the monitor include commands to display and alter the contents of memory and I/O ports, to begin execution at a given address, and to bootstrap programs from disk. The basic I/O functions of monitor provide driving routines for the built-in CRT display and keyboard input, and the floppy disk controller. In this capacity the monitor is always active, even when application programs like the CP/M disk operating system have control of the CPU.

The following sections of this manual will explain how to use the console monitor commands, what facilities are provided by the resident I/O handlers, and how to interface applications programs to the monitor.

XEROX 820 MONITOR COMMAND SUMMARY

The Xerox 820 monitor enters the command mode after it has initialized the system following a power-on or a reset. The following sign-on message is displayed on the console output device as an indication that the monitor is ready to accept commands.

```
... XEROX 820 ...  
Enter A for BOOT  
Enter T for TYPEWRITER  
*  
-
```

Commands consist of a single character command name and zero to three hexadecimal numeric parameters separated by commas or spaces. The command line may be entered using upper case or lower case letters. A carriage return is used as the terminator. Errors within a line can be corrected by typing control-H to delete the last character or control-X to delete the entire line. If a line is entered with an unknown command name, an invalid number or parameters or an out-of-range parameter, an error message will be displayed and the command will not be executed.

The user may wish to halt long running commands like the memory dump before they are finished. This can be done by typing carriage return while the command is doing output. Output can also be frozen temporarily and then re-started by typing repeatedly on the space bar.

The following table summarizes the monitor's command set. The items enclosed in angle brackets represent the numeric parameters expected by the command. A detailed description of each command is provided in the following pages.

<u>command</u>		<u>format</u>
d(ump)	...	D (start), (end)
m(emory)	...	M (address)
t(est)	...	X (start), (end)
f(ill)	...	F (start), (end), (constant)
c(opy)	...	C (source_start), (source_end), (dest_start)
g(oto)	...	G (address)
r(ead)	...	R (unit), (track), (sector)
a(boot)	...	A(boot) from A drive
t(ypewriter)	..	T
i(nput)	...	I (port)
o(utput)	...	O (port), (data)

NOTE: All of the Monitor parameters are hexadecimal.

1) DUMP COMMAND

The dump command outputs a tabular display of the contents of memory in hexadecimal and ASCII representation. Each display line has the following format:

```
AAA DD DD DD DD DD DD DD DD DD DD DD DD DD DD DD CCCCCC
```

where AAAA is the starting memory address of the line in hexadecimal, the DD's are the hex values of the 16 bytes of data starting at location AAAA, and the C's are the ASCII characters equivalent to each data byte. Bytes less than 20 hex are replaced in the ASCII portion of the dump by period.

The dump command accepts zero, one or two address parameters. If two addresses are specified, the block of memory between those two locations will be displayed. Entering only one address will display 256 bytes of memory starting at the specified location. Typing 'D' with no parameters will cause the routine to display the 256 byte block of memory starting at the last address displayed by the dump command.

2) MEMORY COMMAND

The memory examine/change command allows the contents of individual memory locations to be read from and written into using the monitor. This command accepts one parameter representing the memory address at which to begin examining data. The display format is as follows:

AAAA DD _

where AAAA is the current memory address and DD is the hexadecimal value of the data in that location. After displaying the contents of a memory location, the routine waits for one of the following items to be input from the console.

- o Typing a carriage return will cause the routine to display the data at the next memory location, with no modification of content.
- o Typing a minus sign will have a similar effect, except the address is decremented instead of incremental.
- o Typing a two digit hexadecimal number will cause that number to be stored at the displayed address. The new data is stored as soon as the second digit is entered, with no terminating character required.
- o Typing any character other than carriage return, a minus sign or a hexadecimal digit will cause the command to terminate.

3) TEST COMMAND

This command allows the user to test memory for errors caused by defective 16K memory chips, solder bridges and various other problems. Any portion of memory may be tested except the area reserved for the monitor (F000 to FFFF hex). Two parameters are required from the user; the starting address and ending address of the memory block to be tested. Only the high order 8 bits of the addresses entered are actually used however, due to a characteristic of the test algorithm begin employed. If no errors occur, the test routine will output a plus sign every time a test pass is done. A total of 256 plus signs must be output for all possible test patterns to have been tried. When errors are detected an error line will be output in the following format:

AAAA DD should=XX

where AAAA is the address of a location that fails to test, DD is the data read back from the location, and XX is the test pattern that was written there.

4) FILL COMMAND

The fill command allows blocks of memory to be filled with a fixed data constant. Three parameters are required in the command line; a starting memory address, an ending address and a fill constant. Each location in the specified block of memory has the constant written into it and then read back again to check for memory errors. An error line like the one described for the 'X' command is printed for any locations that fail to verify.

5) COPY COMMAND

The copy command allows blocks of data to be moved around in memory. Three parameters are required in the command line; a starting memory address, an ending address, and a destination address. The contents of the block of memory bounded by the first two addresses is copied to the block starting at the third address. As with the fill command, a test is made to verify that each byte of the destination block, when read back, is the same as the corresponding byte in source block.

8) GO TO COMMAND

The goto command allows control of the CPU to be passed to another program by the monitor. This command requires a single parameter from the user representing the address at which to begin execution. The monitor actually passes control to the specified location by executing a CALL instruction. This makes it possible for the external routine to return to the monitor by doing a RET, assuming it does not re-load the stack pointer and loose the return address to the monitor.

9) READ COMMAND

The read command allows individual disk sectors to be read into memory and displayed on the console. Three parameters are required; a drive unit number (range 0 to 1), a track number (range 0 to 27 for 5.25" disks or range 0 to 4C for 8" disks) and a sector number (range 1 to 12 for 5.25" disks or range 1 to 1A for 8" disks). The command routine performs a drive select, track seek and sector read sequence using the supplied parameters. If no errors occur, the contents of the input buffer will be dumped out the 'D' command format. In the event of a disk error, a diagnostic message will be printed in the following format:

disk error XX UAA TBB SCC

where XX represents the 1771 disk controller error status code, AA is the unit number, BB is the track number, and CC is the sector number. The error code is composed of eight bits of status information as described in the table below:

<u>bit</u>	<u>read/write</u>	<u>seek/restore/select</u>
7	drive not ready	drive not ready
6	write protected	unused
5	write fault	unused
4	record not found	seek error
3	crc error	crc error
2	lost data	cannot restore
1	unused	unused
0	always=1	always=0

The least significant bit (LSB) of the error code indicates which of the above sets of error conditions is applicable. If the LSB=1 the disk error was generated by a read or write operation, otherwise it was caused by a seek, restore, or select operation.

10) **BOOT COMMAND**

The boot command command is used to load and begin execution of a one sector long bootstrap loader from the first sector on drive unit zero. The most common use of this command will be to boot up the CP/M disk operating system, although it is not necessarily restricted to this purpose only.

The boot works by reading the contents of track 0, sector 1 into memory at location 80 hex and the jumping to that address to start execution of the code just read in. Normally the routine on sector 1 will be a small loader that in turn reads in a larger program such as the operating system. This two level bootstrap process makes the boot command more application independent. The only requirements are that the first sector of the boot diskette be reserved for a loader and that the bottom 256 bytes of memory are not written over by the program being loaded.

11) **TYPEWRITER COMMAND**

This command allows the XEROX 820 to be used as a standard electronic typewriter. All key strokes will be typed directly on the attached printer in a direct print mode, without displaying any typed information on the screen.

12) **INPUT COMMAND**

This command allows the contents of input ports to be read from using the monitor. It operates very much like the memory examine command, except that input ports are being examined instead of memory locations. A single parameter representing a port number is expected in the command line. The contents of adjacent ports can then be examined by typing carriage return or a minus sign as in the 'M' command. Typing any other key will cause the routine to terminate.

13) **OUTPUT COMMAND**

The output command is provided to allow output ports to be written to using the monitor. Two parameters are expected in the command line; a port number and a data byte to be output to that port. Both parameters should be between 0 and FF hex. After outputting the specified data to the port, this routine simply returns to the monitor instead of stepping to the next location like the input command. This makes it possible to use the output command to initialize Z-80 peripheral devices like the SIO, PIO and CTC.

MONITOR RESIDENT I/O DRIVER FUNCTIONS

This section describes the facilities available in the XEROX 820 monitor for controlling the input/output resources of the XEROX 820 single-board computer.

1) INTERRUPT PROCESSING

The XEROX 820 monitor takes advantage of the powerful interrupt handling capabilities of the Z-80 microprocessor. Interrupts are utilized in the I/O drivers for the console keyboard input, the real-time clock and the floppy disk controller. All necessary initialization tasks and interrupt service routines for these devices are contained in the monitor.

For the most part, the operation of the interrupt mechanism should be transparent to most applications programs that will run under XEROX 820. A few precautions must be taken however, to insure that user written software does not adversely effect the operation of the system. The following list describes the major hazards to the interrupt system;

- o Interrupts should not be disabled permanently by user code, as this will lock-up the console input and real-time-clock routines.
- o The Z-80 'I' register should never be altered. Doing so is GUARANTEED to crash the system.
- o The CPU operates in Z-80 interrupt mode 2 and should not be switched to either of the other two interrupt modes.
- o Adequate stack space must be reserved in user programs to allow at least one level of stack for interrupt return addresses. Use of the stack pointer for 'trick' programming purposes is highly discouraged for the same reason.

The monitor initializes the Z-80 'I' register to point to the system interrupt vector table at location FF00 to FF1F hex. This table contains pre-assigned vector locations for all the peripheral devices on XEROX 820, including those that are not used by any built-in functions in XEROX 820. The devices that are not currently used include SIO channel A, the general purpose PIO and CTC channels 0 and 1. These ports can be initialized and used as needed without affecting the overall system operation. Consult the monitor variables table at the end of this manual for the vector addresses.

2) MEMORY MAPPED VIDEO DISPLAY

The XEROX 820 single-board computer is equipped with a built-in 80 character by 24 line CRT display controller, for use with an external video monitor as the system console output device. The refresh memory for the CRT is bank switchable from the system's 64K byte memory space and includes a hardware address translation circuit for high speed scrolling.

The XEROX 820 monitor contains an output driver routine for the CRT that emulates the characteristics of a typical stand-alone video terminal. An operational summary of the CRT driver is given on the next page.

SYSTEM COMPONENTS

- o All character codes between 20 and 7F hex are directly displayable on the screen. Characters are formed in a 5x7 dot matrix.
- o New characters are stored on the screen at the location occupied by the cursor. The cursor is then moved one place to the right.
- o If the cursor must appear at a screen position occupied by a non-blank character, the presence of the cursor will be indicated by making the overlaid character blink on and off.
- o If the displayable character is output when the cursor is in the right-most column of the screen, an automatic carriage return and linefeed is generated afterwards.
- o If a linefeed in output when the cursor is on the bottom line of the screen, the entire display is scrolled up one line and a new blank line is created on the bottom.
- o All character codes between 00 and 1F hex are interpreted as control characters. Only 12 of these codes have some effect on the CRT display, and are described in the table below. The remaining 20 are treated as nulls.

<u>CODE</u>	<u>KEY</u>	<u>NAME</u>	<u>EFFECT</u>
00	@	NUL	- NONE
01	A	SOH	- NONE
02	B	STX	- NONE
03	C	ETX	- NONE
04	D	EOT	- NONE
05	E	ENQ	- NONE
06	F	ACK	- NONE
07	G	BEL	- NONE
08	H	BS	- BACKSPACE OR CURSOR LEFT
09	I	HT	- HORIZONTAL TAB
0A	J	LF	- LINEFEED OR CURSOR DOWN
0B	K	VT	- CURSOR UP
0C	L	FF	- CURSOR RIGHT
0D	M	CR	- CARRIAGE RETURN
0E	N	SO	- NONE
0F	P	SI	- NONE
10	P	DLE	- NONE
11	Q	DC1	- CLEAR TO END OF SCREEN
12	R	DC2	- NONE
13	S	DC3	- NONE
14	T	DC4	- NONE
15	U	NAK	- NONE
16	V	SYN	- NONE
17	W	ETB	- NONE
18	X	CAN	- CLEAR TO END OF LINE
19	Y	EM	- NONE
1A	Z	SUB	- CLEAR SCREEN
1B		ESC	- ESCAPE SEQUENCE
1C		FS	- NONE
1D		GS	- NONE
1E		RS	- HOME CURSOR
1F	-	VS	- DISPLAY CONTROL CHARACTER

Backspace:

Moves the cursor to the next column to the left. If the cursor is in the leftmost column of the screen, this character has no effect.

Tab:

Moves the cursor right to the next tab stop. The tab stops are fixed at every eighth column, starting from the left.

Linefeed:

Moves the cursor down one line on the screen. If the cursor is the bottom-most line, the screen is scrolled up and a blank line is created on the bottom. The top line is lost.

Cursor Up:

Moves the cursor up one line on the screen. If the cursor is on the top of the screen it rolls around to the bottom.

Cursor Right:

Moves the cursor to the next column to the right. If the cursor is in the rightmost column, there is no effect.

Carrier Return:

Moves the cursor to the leftmost column of the screen.

Clear to EOS:

Clears the contents of the screen from the current cursor position to the end of the bottom line.

Clear to EOL:

Clears the contents of the line the cursor is on, from the cursor position to the end of the line.

Clear Screen:

Clears the entire screen regardless of the current cursor position and places the cursor in the top left corner of the screen. Also re-initializes the CRT driver subroutine.

Escape:

Used to initiate an XY cursor positioning sequence. The cursor can be moved to an arbitrary location on the screen by outputting a 4 character sequence composed of 1) escape, 2) equals sign, 3) row# character and 4) column# character. The row/column number characters are formed by taking the desired row# (range=0 to 23) or column# (range=0 to 79) and adding 20 hex (32 decimal) to it.

Home Cursor:

Moves the cursor to the top left corner of the screen, without altering any characters on the display.

Display CC's:

Functions as a prefix character to force the output of symbols in the character generator that corresponds to ASCII control characters. The next character output to the CRT after outputting a 1F hex will be displayed on the screen regardless of its numeric value. If multiple symbols are to be displayed, 1F must precede each character.

DISPLAY CHARACTER CODES

This table shows the code for each character to be displayed by the XEROX 820. Each character is defined by a unique eight bit code which is represented by a hexadecimal code 'XY' where X represents the 4 most significant bits of the code and Y represents the 4 least significant bits of the code.

There are a total of 128 characters in the font set. Therefore, Y represents a hexadecimal number from 0 to F, and X represents a hexadecimal number from 0 to 7. Therefore, the complete font set is defined by codes from 00 to 7F. If the most significant bit of the eight bit code is set to '1', then the complete font set is duplicated with the blink attribute set. The blinking set of characters is then defined by codes from 80 to FF.

X \ Y	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	□	¢	■		§	½	¼	±	↵	↑	↓	→	←	⌘	®	↔
1	³	²	◊	—	↯	¶	±	μ	†	‡	↵	↵	©	→	↵	
2		!	"	#	\$	%	&	'	()	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	Ⓐ	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[\]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{		}	~	™

KEY STATION NUMBERING

1	2	3	4	5	6	7	8	9	10	11	12	13	14
20	21	22	23	24	25	26	27	28	29	30	31	32	
38	39	40	41	42	43	44	45	46	47	48	49	50	
56	57	58	59	60	61	62	63	64	65	66	67		
73	74										75		

15	16	17	18	19
33	34	35	36	37
51	52	53	54	55
68	69	70	71	72

KEY CODES

KEY NAME	KEY #	UNSHIFTED	SHIFTED	CONTROL
HELP	01	1E	1E	9E
1	02	31	21	91
2	03	32	40	92
3	04	33	23	93
4	05	34	24	94
5	06	35	25	95
6	07	36	5E	96
7	08	37	26	97
8	09	38	2A	98
9	10	39	28	99
0	11	30	29	90
MINUS	12	2D	5F	1F
EQUAL	13	3D	2B	9A
BACKSPACE	14	08	08	88
DELETE	15	7F	7F	FF
- (PAD)	16	2D	2D	AD
7 (PAD)	17	37	37	B7
8 (PAD)	18	38	38	B8
9 (PAD)	19	39	39	B9
TAB	20	09	09	89
Q	21	71	51	11
W	22	77	57	17
E	23	65	45	05
R	24	72	52	12
T	25	74	54	14
Y	26	79	59	19
U	27	75	55	15

Key Codes
(continued)

KEY NAME	KEY #	UNSHIFTED	SHIFTED	CONTROL
I	28	69	49	09
O	29	6F	4F	0F
P	30	70	50	10
[31	5B	7B	1B
]	32	5D	7D	1D
ESC	33	1B	1B	9B
+ (PAD)	34	2B	2B	AB
4 (PAD)	35	34	34	B4
5 (PAD)	36	35	35	B5
6 (PAD)	37	36	36	B6
LOCK	38	---	FUNCTION KEY	---
A	39	61	41	01
S	40	73	53	13
D	41	64	44	04
F	42	66	46	06
G	43	67	47	07
H	44	68	48	08
J	45	6A	4A	0A
K	46	6B	4B	0B
L	47	6C	4C	0C
SEMICOLON	48	3B	3A	7E
APOSTROPHE	49	27	22	60
RETURN	50	0D	0D	8D
LINEFEED	51	0A	0A	8A
UP ARROW	52	01	01	81
1 (PAD)	53	31	31	B1
2 (PAD)	54	32	32	B2

Key Codes
(continued)

KEY NAME	KEY #	UNSHIFTED	SHIFTED	CONTROL
3 (PAD)	55	33	33	B3
L SHIFT	56	---	FUNCTION KEY	---
Z	57	7A	5A	1A
X	58	78	58	18
C	59	63	43	03
V	60	76	56	16
B	61	62	42	02
N	62	6E	4E	0E
M	63	6D	4D	0D
COMMA	64	2C	3C	1C
PERIOD	65	2E	3E	7C
SLASH	66	2F	3F	5C
R. SHIFT	67	---	FUNCTION KEY	---
L. ARROW	68	04	04	84
D. ARROW	69	02	02	82
R. ARROW	70	03	03	83
0 (PAD)	71	30	30	B0
. (PAD)	72	2E	2E	AE
L. CTRL	73	---	FUNCTION KEY	---
SPACE BAR	74	20	20	00
R. CTRL	75	---	FUNCTION KEY	---

3) PARALLEL KEYBOARD INPUT

A parallel keyboard interface is provided on XEROX 820 for systems that will use the built-in keyboard and CRT display as the console I/O device. This interface is designed to connect to an ASCII encoded keyboard with 8 bits of parallel data and a key-pressed strobe.

The XEROX 820 monitor contains an interrupt driven input handler for the keyboard that maintains a 16 character deep FIFO buffer for input data. This makes it possible to do a considerable amount of typing ahead without any input characters being lost. If characters are typed while disk access is going on, they may be lost because the disk routines lock out all lower priority interrupts. Any characters received when the FIFO is full will also be lost.

4) FLOPPY DISK CONTROLLER

The system is equipped with an on-board floppy disk interface capable of controlling up to two Shugart compatible 8 inch drives, or two 5 $\frac{1}{4}$ " drives. The interface hardware is based on a Western Digital 1771 disk controller chip, along with extra TTL support circuitry to provide buffering, drive select, head load timing and data separator functions.

The XEROX 820 monitor contains a complete I/O driver package for the disk controller. Linkage to the disk I/O routines in the monitor is provided by a set of subroutine entry points described later in this manual. The basic functions available are: drive select, restore, seek, track, read sector, and write sector. The user can also specify the track-to-track seek stepping rate, and the sector record length.

All disk functions are verified upon completion, with the final status being returned in the A register. If the command was executed successfully, then A will contain all zeros on return, otherwise it will contain an error status byte as described above under the console monitor 'R' command. The disk driver routines will attempt to recover from any disk I/O errors that occur, so it is generally not necessary for user written programs to try to re-execute commands that fail the first time.

DISK FORMAT

The XEROX 820 is equipped with two (2) compatible Shugart SA400L (5 $\frac{1}{4}$ " drives, or two (8") Shugart SA800 drives. The disks for the SA400L (5 $\frac{1}{4}$ " drives are initialized in a CROMEMCO format, and the disks for the SA800 (8") disks are initialized in an IBM 3740 format.

A format is divided into three (3) parts, field A, field B, and field C. Field A is written at the start of each track known as the preamble. Field B is written once for each sector which consists of a gap between sectors, ID fields, and a data field. Field C is written at the end of each track and is known as a postamble.

The XEROX 820 disks are initialized in the following formats :

Parameter	8"	5 1/4"
Tracks	77	40
Sectors	26	18
Bytes/Sector	128	128
# of Reserved Track for OS	2	3
Disk Capacity	241K	81K

5 1/4" Format

	Number of Bytes	Hex Value of Bytes	Comment
Field A -	16	FF	Preamble on Gap 4A
	4	00	Gap 3
	1	FE	ID Address Mark
	1	XX	Track #
	1	00	
	1	XX	Sector #
	1	00	
*Field B -	1	F7	Generate CRC
	11	FF	Gap 2
	6	00	
	1	FB	Data Address Mark
	128	E5	Data Field 'E5' Data
	1	F7	Generate CRC
	8	FF	Gap 8
Field C -	101	FF	Postamble Gap 4B

8" Format

	Number of Bytes	Hex Value of Bytes	Comment
Field A -	28	FF	Preamble - Write at the start of each track
	6	00	
	1	FC	
	26	FF	
	6	00	Gap 3
	1	FE	ID Address Mark
	1	XX	Track #
	1	00	
	1	XX	Sector #
	1	00	
*Field B -	1	F7	Generate CRC
	11	FF	Gap 2
	6	00	
	1	FB	Data Address Mark
	128	E5	Data Field '5' Data
	1	F7	Generate CRC
	27	FF	Gap 3
Field C -	247	FF	Postamble Gap 4B

* Repeated for one (1) number of sectors per track.

5) SERIAL INPUT/OUTPUT

The XEROX 820 single-board computer has provisions for two completely independent RS-232 serial ports that can be used to interface to printers, CRT terminals and data communications equipment.

6) REAL TIME CLOCK

The XEROX 820 single board computer has Z-80 CTC device that can be used to generate the timebase for interrupt driven timers, real-time clocks, and other time keeping functions. The monitor will initialize CTC channels 2 and 3 to interrupt the processor once a second. Channels 0 and 1 of the CTC are not initialized and can be used for other purposes.

The one second interrupt from the CTC is utilized by the monitor's disk I/O routines to implement the disk motor turn-off function.

7) PARALLEL I/O OPTION

A Z-80 PIO chip has been included on the XEROX 820 for general purpose I/O interfacing. This device is completely unused by any built-in functions. The PIO contains two independent 8 bit parallel I/O ports that can be used to interface to printers, ROM programmers, analog converters, other computers, or just about anything else imaginable. Those interested in using the PIO should consult the schematic drawings for any needed hardware interfacing details. Data about programming the PIO can be found in most Z-80 applications manual.

USER ACCESSIBLE MONITOR ROUTINES AND VARIABLES

This section gives the locations and calling sequences of the user accessible I/O routines in the XEROX 820 monitor. It also describes a number of important monitor variables that may need to be accessed by user written programs.

XEROX 820 subroutines are accessed via a table of JUMP instructions beginning at memory location F000 hex. All monitor calls should be made to these entry points, since the actual addresses of the routines inside XEROX 820 will vary between different releases. Parameter passing conventions for the monitor fall into one of two groups. The character oriented I/O routines all pass data using the A register, while the disk routines pass parameters in C and HL and return status information in A.

Storage for the monitor's stack and working variable occupies the top 256 bytes of memory, from FF00 to FFFF hex. The mode 2 interrupt vector table takes up the first 32 bytes of this block and the stack starts at the very top. In between, are variables used by the monitor resident I/O drivers and interrupt service routines, some of which are described below. Programs should not attempt to write into any locations in this block that are not specifically mentioned below.

1) XEROX 820 SUBROUTINE ENTRY POINTS

F000	ENTO:	JP	INIT	;XEROX 820 cold start entry
F003	ENT1:	JP	PROMPT	;XEROX 820 warm start entry
F006	ENT2:	JP	CONST	;console input status test
F009	ENT3:	JP	CONIN	;console input
F00C	ENT4:	JP	CRTOUT	;memory-mapped CRToutput
F00F	ENT5:	JP	CRTOUT	;memory-mapped CRToutput
F012	ENT6:	JP	SIOST	;SIO port B input status test
F015	ENT7:	JP	SIOIN	;SIO port B input
F018	ENT8:	JP	SIOOUT	;SIO port B output
F01B	ENT9:	JP	SELECT	;disk drive select
F01E	ENT10:	JP	HOME	;restore to track 0
F021	ENT11:	JP	SEEK	;seek track
F024	ENT12:	JP	READ	;read sector into memory
F027	ENT13:	JP	WRITE	;write sector from memory

<u>FUNCTION</u>	<u>PARAMETERS</u>	<u>DESCRIPTION</u>
COLD ...	IN: none OUT: does not return	Perform cold start initialization of XEROX 820 monitor and enter command mode.
WARM ...	IN: none OUT: does not return	Enter XEROX 820 monitor command mode with no re-initialization.
CONST ...	IN: none OUT: status in A	Test for data ready in console input FIFO and return status in A. If data is available then A=FF, else A=00 hex.
CONIN ...	IN: none OUT: character in A	Return character from console input FIFO in A. If FIFO is empty then loop until character is input.
CONOUT ..	IN: character in A OUT: none	Output character passed in A to the memory mapped CRT display
CRTOUT ..	IN: character in A OUT: none	Output character passed in A to the memory-mapped CRT display.
SIOST ..	IN: none OUT: status in A	Test for received data available from SIO channel B and return status in A. if data is available then A=FF, else A=00 hex.
SIOIN ...	IN: none OUT: character in A	Return received data from SIO channel B in A. Loop until data is received if none is available on entry.
SIOOUT ...	IN: character in A OUT: none	Output character passed in A to SIO channel B transmit register.
SELECT ...	IN: unit number in C OUT: status in A	Select specified drive for future restore, seek, read or write command. If the drive is not ready, then the currently selected drive is left on.
HOME ...	IN: none OUT: status in A	Move read/write head to home position at track 0 and verify if it got there.
SEEK ...	IN: track number in C OUT: status in A	Move read/write head to specified track and verify if it got there.
READ ...	IN: sector number in C buffer pointer in HL OUT: status in A	Read specified sector on current track into memory data buffer.

WRITE . . . IN: sector number in C Write specified sector on current
 buffer pointer in HL Track from memory data buffer.
 OUT: status in A

2) STORAGE ALLOCATION FOR XEROX 820 VARIABLES

(INTERRUPT VECTOR TABLE)

FF00 SIOVO:	DEFS2	;SIO port B xmit buffer empty
FF02 SIOV1:	DEFS2	;SIO port B external/status change
FF04 SIOV2:	DEFS2	;SIO port B receive data available
FF06 SIOV3:	DEFS2	;SIO port B special receive condition
FF08 SIOV4:	DEFS2	;SIO port A xmit buffer empty
FF0A SIOV5:	DEFS2	;SIO port A external/status change
FF0C SIOV6:	DEFS2	;SIO port A receive data available
FF0E SIOV7:	DEFS2	;SIO port A special receive condition

FF10 CTCVO:	DEFS2	;CTC channel 0 interrupt
FF12 CTCV1:	DEFS2	;CTC channel 1 interrupt
FF14 CTCV2:	DEFS2	;CTC channel 2 interrupt
FF16 CTCV3:	DEFS2	;CTC channel 3 interrupt

FF18 SYSVA:	DEFS2	;system PIO port A interrupt
FF1A SYSVB:	DEFS2	;system PIO port B interrupt

FF1C GENVA:	DEFS2	;general purpose PIO port A interrupt
FF1E GENVB:	DEFS2	;general purpose PIO port B interrupt

(CONSOLE KEYBOARD INPUT VARIABLES)

FF20 FIFO:	DEFS16	;input data fifo buffer
FF30 FIFCNT:	DEFS1	;number of characters in FIFO
FF33 LOCK:	DEFS1	;character used for software shift

Theory of Operation — CENTRAL PROCESSOR

CLOCK GENERATOR:

All the system clocks with the exception of the baud clock and the video dot clock are generated from a master oscillator operating at 20 Mhz. The 20 Mhz clock is scaled by the divide-by-5 section of decade counter to provide 4 Mhz for use in the floppy disk data separator. The 2 Mhz clock for the disk controller is generated from the 4 Mhz clock by the remaining divide by two section.

The 2.5 Mhz processor clock is generated by dividing the master 20 Mhz clock by 8 with binary counter. The column address strobe "CAS", and the address multiplexer control "MUXC", are derived from the 20 Mhz clock. When memory request "MREQ" is low and refresh "RFSH" is high generation of "CAS" and "MUXC" is enabled. "RFSH" disables the generation of "CAS" and "MUXC" by holding shift register reset. This is done to take advantage of the low power row address strobe "RAS" only refresh mode of the 16K dynamic RAM.

RESET CONTROLLER:

Two types of reset take place on the board. Power on reset is detected and conditioned by part of hex schmitt inverter. The pushbutton reset is also conditioned by a part of hex schmitt inverter.

PORT ADDRESS DECODING:

Octal decoder is used to select the appropriate I/O device based on the binary value of the address bits A2, A3, & 4. When A7 is low and "M1R" is high, a low on "IORQ" will cause the appropriate output of the decoder to go low, selecting the I/O device for a read or write operation.

PORT 0-3 = CHANNEL A BAUD RATE (WRITE ONLY)
PORT 4 = SIO CHANNEL A DATA
PORT 5 = SIO CHANNEL B DATA
PORT 6 = SIO CHANNEL A CONTROL
PORT 7 = SIO CHANNEL B CONTROL
PORT 8 = GP PIO PORT A DATA
PORT 9 = GP PIO A CONTROL
PORT A = GP PIO PORT B DATA
PORT B = GP PIO PORT B CONTROL
PORT C-F = CHANNEL B BAUD RATE (WRITE ONLY)
PORT 10 = 1771 STATUS/COMMAND REGISTER
PORT 11 = 1771 TRACK REGISTER
PORT 12 = 1771 SECTOR REGISTER
PORT 13 = 1771 DATA REGISTER
PORT 14-17 = CRT SCROLL REGISTER (WRITE ONLY)
PORT 18 = CTC CHANNEL 0
PORT 19 = CTC CHANNEL 1
PORT 1A = CTC CHANNEL 2
PORT 1B = CTC CHANNEL 3
PORT 1C = SYSTEM DATA PORT
PORT 1D = SYSTEM CONTROL PORT
PORT 1E = KEYBOARD DATA PORT
PORT 1F = KEYBOARD CONTROL PORT

SYSTEM COMPONENTS

DISK TRANSFER SYNCHRONIZATION:

In order to successfully execute the high speed data transfers between the processor and the disk controller, the fast Z-80 non maskable interrupt "NMI" response was employed. During reads and writes to and from the disk controller, the data at memory location 66 hex is retrieved and stored. This location is overwritten with a RETURN instruction. After this setup is accomplished the processor executes a HALT instruction. When the processor is in a HALT condition, a DATA REQUEST (DRQ) or an INTERRUPT REQUEST (IRQ) from the disk controller will cause a non-maskable interrupt to be generated. The processor then executes the RETURN instruction at 66 hex and returns to transfer the data to or from the disk controller. When the 128 byte transfer is complete the old data is restored and the processor resumes normal operation. This hardware assistance obviated the necessity for a DMA device by eliminating the disk controller "DRQ" status test.

CRT DISPLAY CONTROLLER

VIDEO SCROLLING:

In order to eliminate the delay associated with software scrolling, hardware assistance was employed. For ease of understanding, the CRT RAM resides from 3000 hex to 3FFF hex. Writing into the scroll register adds an offset to the line address developed by the line counter. The net effect is similar to the rotation of a cylinder whose axis is horizontal and perpendicular to the line of shift. The amount of rotation is determined by the magnitude of the number contained in the scroll register. For instance, an offset of zero puts the data at location 3000 hex (of the CRT memory) at the bottom of the screen. If the offset was one, the data at 3000 hex would be displayed on the line next to the bottom. An offset of seventeen hex (23 decimal) puts the data at location 3000 hex at the top of the screen.

VIDEO RAM ADDRESSING:

If the processor is doing a read or write to video RAM "CRTCE" (CRT memory access enable) will go low. When "CRTCE" goes low, the address from the processor is selected instead of the address generated by the counter chain. This gives the processor access to the video RAM for read or write operations. CPU ACCESS OF VIDEO RAM:

During read or write operations involving the video RAM and the CPU, "CRTCE" will go low. When "CRTCE" goes low the processor address bus is selected as the address source for the video RAM. A low on "CRTCE" is also used as a term in the direction control logic for data buss access. During a processor read operation, data from the video RAM at the specified address is allowed onto the processor data bus. During a processor write operation, data from the processor is written to the video RAM at the specified address.

VIDEO GENERATION:

While in the display mode, ASCII data from the video RAM and scan address data are used to select the proper dot patterns from the character generator. The character generator contains 1 Font pattern of 128 characters.

DISPLAY BLANKING:

The display is blanked during horizontal retrace, vertical retrace, CPU access, and decode of scan counts 8 & 9. Blanking is accomplished by disabling the character generator.

64 K RAM AND BANK SWITCHING

REFRESH:

During the refresh cycle, the Z-80 places the refresh address on the lower 7 bits of the address bus. When this address is stable in the RAM array, the "RFSH" pin on the Z-80 goes low. The active low "RFSH" generates a "RAS" on all RAMs. An active "RFSH" disables the generation of both "CAS" and "MUXC".

BANK SWITCHING:

Bit 7 of port 1C hex is the bank switch control. When the output is high, the ROMs and the CRT display appear in the lower 16K block. When bit 7 of port 1C hex is low, all the 64K RAM is available to the processor.

FLOPPY DISK CONTROLLER, SYSTEM PIO, AND CTC

FLOPPY DISK CONTROLLER:

The 1771 Disk Controller performs all the control functions required to interface to a floppy disk drive. The only support required by the 1771 is external data separation, inverting data bus transceivers, head load timer, and buffering to and from the drive(s).

DATA SEPARATOR:

Presettable counter is used as a timing reference developed by the system clock. Raw data coming from the disk drive is used to preload the counter. If the counter does not receive a data bit between clocks the counter in effect times out and presents the controller with a logic zero. If the counter receives data between clocks, the controller will see a logic one on its data input.

DATA BUS BUFFERING:

Inverting transceivers adapt the 1771 to the non-inverted Z-80 data bus. During a read operation, data from the 1771 is allowed onto the processor's data bus. Otherwise, the processor's data bus always drives the 1771's data inputs.

CTC:

The CTC resides at ports 18 hex through 1B hex (see page 10-38). All the inputs and outputs associated with the CTC are available to the user. Refer to the strapping option section for pin assignments.

SYSTEM PIO:

The system PIO resides at ports 1C hex through 1F hex. The "A" side of the system PIO controls the floppy disk drive select, bank switching, disk power switching, sensing keyboard data available (for polled keyboard applications), and on un-committed user definable I/O bit. The bit allocations are as follows:

- BIT 0 = DVSEL 1
- BIT 1 = DVSEL 2
- BIT 2 = SIDE
- BIT 3 IS USED FOR KEYBOARD DATA AVAILABLE
- BIT 4 IS 8"/5¼" DISK SELECT
- BIT 5 ASSIGNED FOR FUTURE USE
- BIT 6 CONTROLS DISPLAY CHARACTER SET
- BIT 7 CONTROLS THE BANK SWITCHING (0=RAM)

The drive select information should be presented to the system PIO in the following manner:

- read port 1C hex to maintain system status data
- mask off the lower 3 bits
- "or" in the desired drive & side information
- write the modified data to port 1C hex

The "B" side of the system PIO is devoted to the keyboard. The keyboard port is eight bits wide and is fully buffered.

GENERAL PURPOSE PIO AND SIO

The G.P. PIO provides the user with 16 bits of user definable input or output or a mix of input and output on nibble boundaries. The G.P. PIO resides at ports 08 hex -0B hex. The PIO will support all modes of interrupt supported by the Z-80. For detailed programming information refer to the Z-80 PIO data sheet. For applications information, refer to the strapping option section.

SIO:

The Z-80 SIO supports two full channels of serial I/O with the capability of supporting full RS-232 protocol on both channels. In addition, the A side of the SIO can provide clocks to synchronous modems or receive clocks from the modem.

Channel A of the SIO can be configured to interface to a modem or a terminal. Refer to the strapping option sheets for detailed instructions. Refer to the SIO data sheet for programming information.

Channel B of the SIO is dedicated for printer operation and has no strapping options.

The "SYNC" pins on both channels of the SIO have been connected to the Rx data pins to facilitate baud rate selection. Utilizing this feature, the start bit duration can be timed, and the baud rate can be set accordingly.

BAUD RATE GENERATOR:

The COM 8116 provides the user with two programmable baud rate generators. Channel A baud rate resides at port 00 hex and is write only. Channel B baud rate resides at port 0C hex and is also write only. The programming procedure is as follows:

00 hex =	50 Baud
01 hex =	75 Baud
02 hex =	110 Baud
03 hex =	134.5 Baud
04 hex =	150 Baud
05 hex =	300 Baud
06 hex =	600 Baud
07 hex =	1200 Baud
08 hex =	1800 Baud
09 hex =	2000 Baud
0A hex =	2400 Baud
0B hex =	3600 Baud
0C hex =	4800 Baud
0D hex =	7200 Baud
0E hex =	9600 Baud
0F hex =	19.2 Kbaud

INTERRUPT STRUCTURES:

All the Z-80 family devices on this board are capable of supporting mode 0, 1, and 2 interrupts. Mode 2 interrupts are used in the monitor delivered with the system. The I register in a unmodified system is loaded with 0FF hex. The priority chain is organized high to low as follows:

- SIO CHANNEL A
- SIO CHANNEL B
- SYSTEM PIO PORT A
- SYSTEM POI PORT B
- GP PIO PORT A
- GP PIO PORT B
- CTC CHANNEL 0
- CTC CHANNEL 1
- CTC CHANNEL 2
- CTC CHANNEL 3

			J8	
			1	2
all odd # pins are grounded			o	o port A STROBE
			o	o port A READY
			o	o port A bit 0
			o	o port A bit 1
			o	o port A bit 2
			o	o port A bit 3
			o	o port A bit 4
			o	o port A bit 5
			o	o port A bit 6
			o	o port A bit 7
			o	o port B READY
			o	o port B STROBE
			o	o port B bit 0
			o	o port B bit 1
			o	o port B bit 2
			o	o port B bit 3
			o	o port B bit 4
			o	o port B bit 5
			o	o port B bit 6
		o	o port B bit 7	
		39	40	

	J10		
	2	1	
SYSTEM CLOCK	0	0	CLOCK/TRIGGER 0
ZC-TO0	0	0	CLOCK/TRIGGER 1
ZC-TO1	0	0	CLOCK/TRIGGER 2
ZC-TO2	0	0	CLOCK/TRIGGER 3
	8	7	

	J7	
	6 1	
	0 0	
	0 0	
6-10 grounded	0 0	Vertical Sync
	0 0	Horizontal Sync
	0 0	Video
	10 5	

SERIAL I/O CONNECTOR PIN ASSIGNMENTS CHANNEL A (J4)

	J4		
	1	14	
PROTECTIVE GROUND	o	o	
TRANSMIT DATA	o	o	TRANSMIT CLOCK
RECEIVE DATA	o	o	
REQUEST TO SEND	o	o	RECEIVE CLOCK
CLEAR TO SEND	o	o	
DATA SET READY	o	o	
PROTECTIVE GROUND	o	o	DATA TERMINAL READY
CARRIER DETECT	o	o	
	o	o	
	o	o	
	o	o	
	o	o	
	o	25	
	13		

SERIAL I/O STRAPPING OPTIONS FOR CHANNEL A (J9)

Only channel A is capable of utilizing baud clocks from an external device or of providing baud clocks to an external device. When providing the baud clock to the external device the SIO must use the same clock source.

	J9		
	1	2	
	o	o	
	o	o	
	o	o	(M) RXD to Pin 3
	o	o	(T) TXD to Pin 2
	o	o	(M) TXD from Pin 2
	o	o	(T) RXD from Pin 3
	o	o	(M) CTS to Pin 5
	o	o	(T) RTS to Pin 4
	o	o	(M) RTS from Pin 4
	o	o	(T) CTS from Pin 5
	o	o	(M) DCD to Pin 8
	o	o	(T) DTR to Pin 20
	o	o	(M) DTR from Pin 20
	o	o	(T) DCD from Pin 8
Clock supplied to Modem as RX Clock	o	o	
Clock supplied to SIO with RX Clock	o	o	
Modem supplies SIO with RX Clock	o	o	
Clock supplied to SIO with TX Clock	o	o	
Modem supplies SIO with TX Clock	o	o	
Clock supplied to Modem with TX Clock	o	o	
	39	40	

DATA SET READY IS ACTIVE ON BOTH CHANNELS

Legend

(M) Indicates modem (Data Communications Equipment) function

(T) Indicates terminal (Data Terminal Equipment) function

For instance, exercising the (T) strap options will allow communication with a modem. Exercising the (M) strap options would allow communication with a terminal.

TXD = Transmitted Data

RXD = Received Data

RTS = Request to Send

CTS = Clear to Send

DTR = Data Terminal Ready

DCD = Data Carrier Detect

SERIAL I/O CONNECTOR PIN ASSIGNMENTS CHANNEL B (J3)

	J3		
	1	14	
Ground	o	o	
Receive Data	o	o	
Transmit Data	o	o	
Clear to Send	o	o	
Request to Send	o	o	
Data Set Ready	o	o	
Ground	o	o	Data Carrier Detect
Terminal Ready	o	o	
	o	o	
	o	o	
	o	o	
	o	o	
	o	25	
	13		

KEYBOARD CONNECTOR PIN ASSIGNMENTS (J2)

	J2	
	1	14
BIT 0	o	o
BIT 1	o	o
BIT 2	o	o
BIT 3	o	o
BIT 4	o	o
BIT 5	o	o
BIT 6	o	o
BIT 7	o	o
STROBE	o	o
	o	o
	o	o
	o	o
	o	o
+5 volts	o	25
	13	

Pins 14-25 are all grounded

DISK DRIVE CONNECTOR PIN ASSIGNMENTS (J1)

	J1	
	1	20
8/5¼ Select	o	o
Index	o	o
Select 1	o	o
Select 2	o	o
Side	o	o
HDLD	o	o
Step In	o	o
Step	o	o
Write Data	o	o
Write	o	o
TRK 00	o	o
Write Protect	o	o
Read Data	o	o
Low Current	o	o
Ready	o	37
	19	

Pins 20-37 are all grounded

(This page intentionally left blank)